Collaborative En Route Airspace Management Considering Stochastic Demand, Capacity, and Weather Conditions

by

Jeffrey M. Henderson

A dissertation presented to the Faculty of Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Civil Engineering

Dr. Antonio Trani, Chair
Dr. Hojong Baik
Dr. Hesham Rakha
Dr. Gerardo Flintsch
Dr. C. Patrick Koelling

March 26, 2008
Blacksburg, Virginia

Keywords: Airspace, Thunderstorms, Equity, Air Traffic Control, Simulation

Collaborative En Route Airspace Management Considering Stochastic Demand, Capacity, and Weather Conditions

Jeffrey M. Henderson

ABSTRACT

The busiest regions of airspace in the U.S. are congested during much of the day from traffic volume, weather, and other airspace restrictions. The projected growth in demand for airspace is expected to worsen this congestion while reducing system efficiency and safety. This dissertation focuses on providing methods to analyze en route airspace congestion during severe convective weather (i.e. thunderstorms) in an effort to provide more efficient aircraft routes in terms of: en route travel time, air traffic controller workload, aircraft collision potential, and equity between airlines and other airspace users. The en route airspace is generally that airspace that aircraft use between the top of climb and top of descent.

Existing en route airspace flight planning models have several important limitations. These models do not appropriately consider the uncertainty in airspace demand associated with departure time prediction and en route travel time. Also, airspace capacity is typically assumed to be a static value with no adjustments for weather or other dynamic conditions that impact the air traffic controller. To overcome these limitations a stochastic demand, stochastic capacity, and an incremental assignment method are developed. The stochastic demand model combines the flight departure uncertainty and the en route travel time uncertainty to achieve better estimates for sector demand. This model is shown to reduce the predictive error for en route sector demand by 20% at a 30 minute look-ahead time period.

The stochastic capacity model analyzes airspace congestion at a more macroscopic level than available in existing models. This higher level of analysis has the potential to reduce computational time and increase the number of alternative routing schemes considered. The capacity model uses stochastic geometry techniques to develop predictions of the distribution of flight separation and conflict potential. A prediction of dynamic airspace capacity is

calculated based on separation and conflict potential.

The stochastic demand and capacity models are integrated into a graph theoretic framework to generate alternative routing schemes. Validation of the overall integrated model is performed using the fast time airspace simulator RAMS. The original flight plans, the routing obtained from an integer programming method, and the routing obtained from the incremental method developed in this dissertation are compared. Results of this validation simulation indicate that integer programming and incremental routing methods are both able to reduce the average en route travel time per flight by 6 minutes. Other benefits include a reduction in the number of conflict resolutions and weather avoidance maneuvers issued by en route air traffic controllers. The simulation results do not indicate a significant difference in quality between the incremental and integer programming methods of routing flights around severe weather.

ACKNOWLEDGEMENTS

I would first and foremost like to thank Liping Fu for his ongoing encouragement to study at the doctoral level. As my Masters advisor he provided me with the positive outlook that gave me confidence in my abilities. Without him I would not have continued my studies.

Antonio Trani continued this positive attitude which made my studies very enjoyable at Virginia Tech. He is a wonderful influence on all young researchers in the way he approaches his work. I will always remember the quote that he considers every day at Virginia Tech a holiday.

I would also like to thank all of the members of the Air Transportation Systems Lab under the guidance of Dr. Trani. They were all instrumental in improving my knowledge of air transportation. More than that they provided friendship and support that I needed to complete my work.

I must acknowledge all of the financial support that I received while studying at Virginia Tech. I thank the Virginia Tech College of Engineering, NASA, and the FAA for their generous support of research.

Lastly, I would like to thank my family for all the love and emotional support that I received during my graduate studies.

DEDICATION

To Kyle.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

The commercial aviation industry is a significant component of the global economy. More than 4.5 percent of the world economic output is associated with the air transport component of civil aviation (ICAO, 1998). Other metrics indicate that 40 percent of world trade, by value, is carried by air transport and the industry as a whole provides 28 million jobs through direct industries and associated indirect economies (Collaborative Forum of Air Transport Stakeholders, 2003). The stated impact of air transportation varies over time, by study, and by funding agency but the importance of the industry in maintaining economic links within and between countries is beyond doubt.

Projected rising demand for air travel has the potential to increase congestion and reduce operational safety (Figure 1). Revenue passenger miles (RPM) returned to pre-September 2001 levels in 2004 with passenger enplanements and available seat miles returning to 2000 levels during 2005 (FAA, 2007). Aircraft traffic at en route centers increased 2.8 percent to 47.5 million in 2006, which exceeds pre-September 2001 levels (FAA, 2007). Furthermore, FAA predicts traffic through en route centers to grow faster than at FAA towered airports. The projected annual increase in en route air traffic is shown in Table 1.

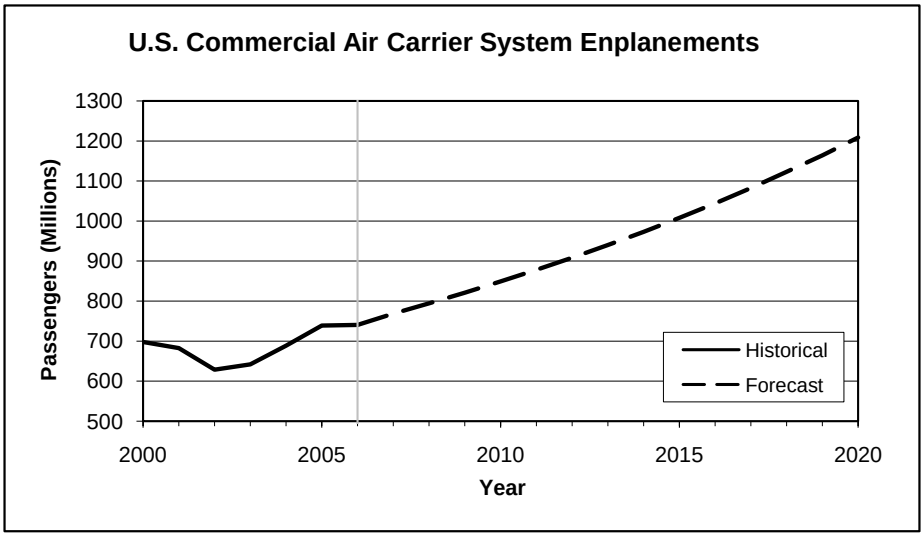

Figure 1: Commercial Aviation Demand Forecast (Data Source: FAA (2007)). Includes both domestic and international passengers.

Table 1: En Route Traffic Growth Projection (Data Source: FAA (2007)).

Year	Total Annual Percentage Growth
2005	2.7%
2006	-2.7%
2007	2.2%
2008	2.6%
2009	2.5%
2010	2.7%
2011-2020	3.3%

Delays to aircraft and passengers continue. A recent Air Travel Consumer Report estimates that only 73.4 percent of scheduled flights arrived on-time (Office of Aviation Enforcement and Proceedings, 2008). An on-time flight arrives to the destination airport within 15 minutes of the published arrival time. Prominent causes of delay were noted as: cancellations (3.5 percent), extreme meteorological conditions (1.4 percent), maintenance and other delays within airline control (9.1 percent), heavy traffic volume and delays due to non-extreme weather (10.4 percent), and previous aircraft arriving late to origin airport (10.9 percent). The percentage of flights arriving on time plus the summation of all flight delay causes will equal 100, however less notable causes are not included here. It is thought that increased demand for air transportation resources will further degrade on-time performance without an appropriate management intervention strategy.

Airlines are shifting their fleet mix to include smaller aircraft, such as regional jets, in an attempt to reduce costs and compete with low-cost carriers. Regional jets carry 39 percent of commercial traffic at the 35 busiest airports in the US, up from 30 percent in 2000 (FAA, 2006a). Regional jets are turbo-fan powered aircraft that have historically seated less than 70 passengers with projections of 70 to 100 seats in the future. These aircraft increase operational complexity and number of aircraft in the National Airspace System (NAS) since to sustain passenger levels more regional jets are required, as compared to mainline jets, to service a route. This trend will further degrade airspace operations that already experience annual increases in congestion. Another factor in future airspace operations is the use of

very light jets for demand-responsive commercial air-taxi service. These jets seat one to four passengers and have the potential to impact operations if allowed to operate in congested airspace. However, the popularity of this service in the future is still largely unknown.

The Federal Aviation Administration within the Department of Transportation (DOT) retains authority for NAS operations. The mission of the FAA is to provide"a safe, secure, and efficient global aerospace system that contributes to national security and the promotion of US aerospace safety" (FAA, 2004a). In support of this mission the FAA has goals of increasing safety and capacity over current NAS operations.

The implementation of information technologies at air traffic control facilities in an effort to improve NAS operations is known as Air Traffic Management (ATM). Potentially, ATM can reduce delays, improve safety, and allow air carriers to fly more efficient routes. The deployment of ATM procedures, generally, is a contentious issue between airlines and the FAA. The problem is that the authority controlling decisions, i.e. the FAA, does not have all of the relevant airline information and may make decisions that are not equitable to all NAS users. Also, airlines have also argued that they should retain as much control as possible over their economic decisions.

Collaborative Decision-Making (CDM) is an initiative that originated from the FAA's airline data exchange program of 1993 (Wambsganss, 1997). CDM is essentially a distributed decision-making environment in which the FAA estimates the capacity of the airspace and defines the operational constraints. These constraints are communicated to airlines, which are able to plan around anticipated disruptive capacity restrictions and communicate the airline's intentions back to the FAA. Dynamic information exchange is a core element of CDM as it allows better information to be available to both the FAA and airlines. As conditions change frequent interaction between the FAA and airlines are required so that all parties can adapt. The decision makers include the FAA, airline companies, and other parties with a stake in operations of the NAS. The rules are established and refined at regular meetings of all the CDM participants.

1.1 Problem Statement

Efficient NAS operations continue to be a concern for the FAA and NAS users. The busiest airspace are congested for significant periods of time due to a combination of traffic volume, weather, and other airspace restrictions. This congestion, which is expected to worsen in the foreseeable future, can lead to reduced safety, which conflicts with FAA's first priority of promoting an accident-free NAS. During near-capacity operating conditions passengers also experience significant delays due to flight rescheduling and flight cancellations. The problem is to achieve the highest levels of efficiency and safety while ensuring that no airline gains a competitive advantage through airspace management decisions.

1.2 Objective

The primary research interest is the management of en route airspace with the objective of reducing delays and improving safety within the CDM framework of equitable decisions for all parties. In this work air traffic control workload is used as a measure of en route airspace safety since the performance of controllers has a direct impact on the potential for accidents. The scope of air traffic management includes dynamic estimation of en route airspace capacity, devising strategies to reduce congestion, and most importantly preventing violations of minimum separation standards between aircraft. Appropriate methods to communicate expected levels of congestion and the FAA's management strategy to NAS users are also under consideration.

1.3 Contribution

Airspace management models are available that address strategic aircraft routing with and without CDM considerations as described in the literature review. However, it is unclear if these models adequately address the problem described. Some models may be too computationally prohibitive for realistic instances of management problems. Additionally, there are noted conceptual flaws in the approaches.

The new approach presented in this dissertation studies airspace operations at a level of detail not yet considered. Currently, only detailed microscopic analysis models or very

macroscopic experienced-based techniques are available to predict system response under increasing levels of traffic load. The new approach models en route operations at a level of detail capable of fast-time system evaluation. This model has the potential to analyze congestion without requiring the use of computationally expensive microscopic techniques.

Stochastic geometry techniques and spatial point processes are used to develop random separation and conflict potential models for the en route airspace. Conflicts are violations of minimum separation standards between aircraft. To the author's knowledge this is the first separation prediction model for the en route airspace. The conflict potential model is able to establish the random rate of conflicts without considering every pair of flights in the nearby airspace, which is the approach of existing models. The random separation and conflict potential models are then used to evaluate the potential impact of a convective weather system on en route airspace capacity.

Further enhancements over existing airspace demand and capacity models more appropriately consider system uncertainty by modeling stochastic elements of airspace demand and capacity. The airspace demand model combines predeparture and en route sources of uncertainty, which are not available in current models, to produce improved estimates of airspace sector demand. Also, the development of a rigorously defined flow constrained area is presented that identifies regions of airspace impacted by the weather or airspace restriction event. The selection of flights to analyze would be difficult without a precise definition for a flow constrained area.

1.4 Organization of Document

The remainder of this document is organized as follows.

- Section 2 provides a literature review on aspects of air traffic management relevant to this dissertation.
- Section 3 describes the validation of an earlier airspace planning model developed at Virginia Tech. Lessons learned in the validation are used to develop the models central to this dissertation.

- Section 4 describes the framework used in this dissertation and how the model components interact to solve the airspace planning problem.
- Section 5 includes a description of the module that estimates demand for en route airspace considering random flight departure time and en route conditions.
- Section 6 includes the supporting models to calculate en route sector capacity considering the impact of convective weather on available airspace and air traffic controllers.
- Section 7 contains a description of the routing procedure that includes: identification of congested airspace and flights operating in that airspace, grouping of flights, and the routing of the flight groups.
- Section 8 includes a description of the Java tool used in computations (AirPlanJ) and the application of the tool to an airspace restriction scenario. The model presented in this dissertation is then validated using an airspace simulator known as RAMS.
- Sections 9 and 10 provide the conclusions and recommendations that follow from the analysis presented in the above sections.

2 LITERATURE REVIEW

This section presents a review of past airspace management techniques and related issues. Aspects of collaborative decision-making, the airspace occupancy and planning models, conflict detection and resolution, and other issues that impact airspace management are discussed. The Center-Tracon Automation System (CTAS) controls flights at the en route and terminal area traffic interface and the relationship between CTAS and the proposed model will be an important consideration.

2.1 National Severe Weather Playbook

The National Severe Weather Playbook (FAA, 2006e) is the FAA's current state-of-practice for re-routing air traffic during severe weather events and other airspace capacity constrained conditions. The Playbook is under constant revision to add additional plays and eliminate less effective strategies. There are approximately 140 plays in the Playbook split approximately 50%/50% between en route scenarios and TRACON (terminal radar approach control) airspace.

An example play from the Playbook is shown in Figure 2 with routing details described in Table 2. Play NO J6 1 is one of three plays for jet route J6, a route oriented northeast/southwest near the southern edge of center ZID and northwest edge of ZDC. The origin field of the routing table (Table 2) for this example only includes centers (e.g. ZBW) but can include airports and NAVAIDs (navigational aids). Similarly, the destination field only includes airports (e.g. DFW) but can also include centers and NAVAIDs. The route field uses a combination of airports, standard instrument departures (SID), jet routes, NAVAIDs, and standard terminal arrivals (STARs) to define the jet routing procedure. The filter field is not used here but can be used to remove a subset of flights from the routing restrictions.

The main criticism of the playbook is the inability to adapt to real-time information (Wanke et al., 2005). The plays are developed and enacted based on the experience of the traffic manager. No mathematical modeling of scenarios is made before a play is used and airlines have limited input on the routing of their flights. Also, there have been issues with standardization of management strategies since, historically, responses to similar severe

Impacted Area or Flow: J6 BETWEEN MRB-BWG

Facilities Included: ZBW/ZNY/ZDC/ZID/ZOB/ZTL/ZME/ZHU/ZJX

Instructions: REROUTE ANY AIRBORNE TRAFFIC AND INTERNAL DEPARTURES VIA THE FOLLOWING ROUTES

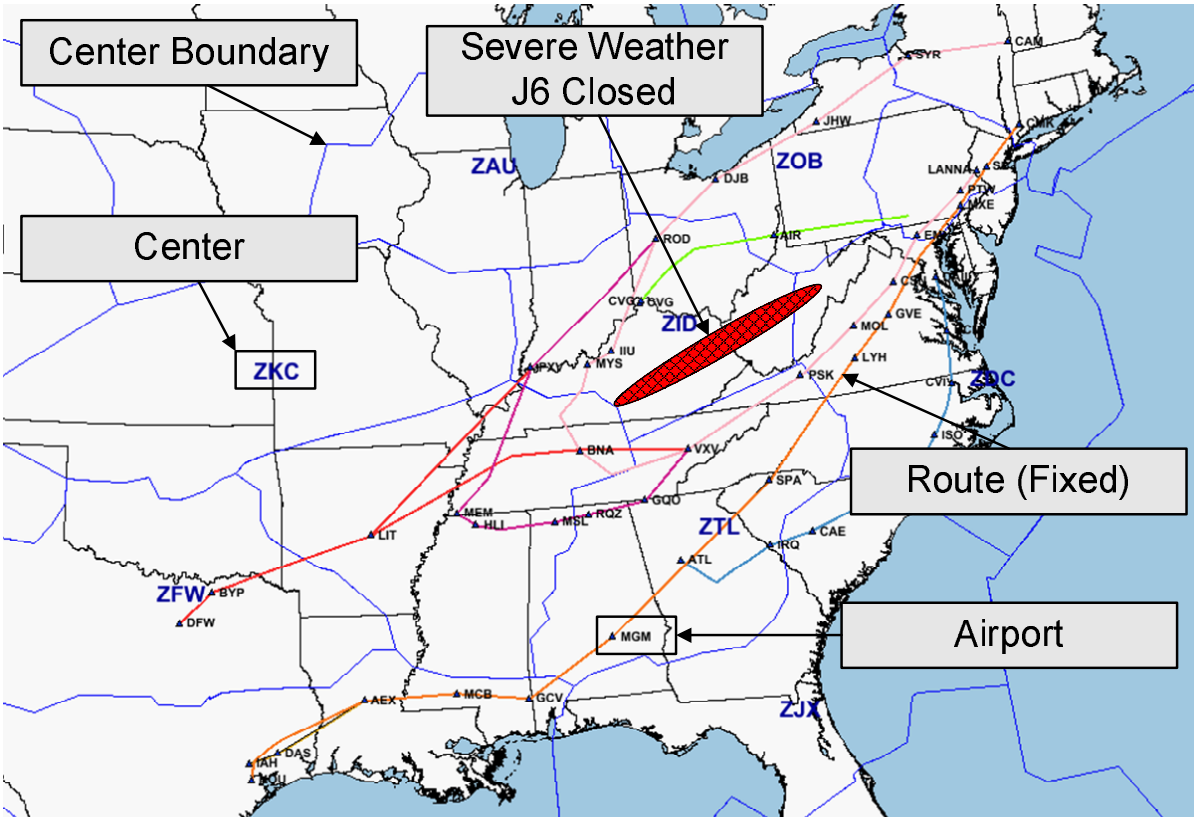

Figure 2: Sample Playbook Entry

Table 2: Sample Routing Procedure.

ORIGIN	ROUTE	DEST
ZBW	CAM J547 SYR J29 ROD J39 IIU MYS HEHAW4	BNA
ZBW	CMK J75 GVE J37 MGM J590 GCV MCB AEX ROKIT7	HOU
ZBW	CAM J547 SYR J29 PXV J131 LIT BYP5	DFW
ZID	PXV J131 LIT BYP5	DFW
ZDC	MOL J22 VXV J46 BNA J46 J6 LIT BYP5	DFW
ZDC	DAILY J61 HUBBS J193 WEAVR J121 J4 IRQ SINCA3	ATL
ZDC	GVE J37 MGM J590 GCV MCB AEX DAS6	IAH
ZDC	GVE J37 MGM J590 GCV MCB AEX ROKIT7	HOU
ZDC	MOL J22 VXV GQO HLI1	MEM
ZDC	MOL J22 VXV VOLLS6	BNA
ZNY	LANNA J48 MOL J22 VXV J46 BNA J46 J6 LIT BYP5	DFW
ZNY	KIPPI J80 AIR CINCE4	CVG
ZNY	MXE J75 GVE J37 MGM J590 GCV MCB AEX DAS6	IAH
ZNY	MXE J75 GVE J37 MGM J590 GCV MCB AEX ROKIT7	HOU
ZNY	LANNA J48 MOL J22 VXV GQO HLI1	MEM
ZNY	LANNA J48 MOL J22 VXV VOLLS6	BNA
ZBW	CMK J75 GVE J37 MGM J590 GCV MCB AEX DAS6	IAH
ZBW	CAM J547 SYR J29 PXV WLDER4	MEM

weather events have been handled differently based on the traffic manager on duty.

2.1.1 Airspace Flow Program (AFP)

The FAA began the Airspace Flow Program (AFP) operational concept in June 2006 (FAA, 2006d). AFP complements the Playbook by reducing flights through congested airspace while retaining the Collaborative Decision-Making philosophy. The iterative AFP process assigns the responsibility of estimating airspace capacity to the FAA (Figure 3). The airlines are then able to substitute departure times of their own flights so that, for example, a higher priority flight is swapped with a lower priority flight to reduce the high priority flight delay.

Airlines retain ownership of slots occupied by cancelled flights but the adaptive compression algorithm may move slots to a later time to ensure all slots are used.

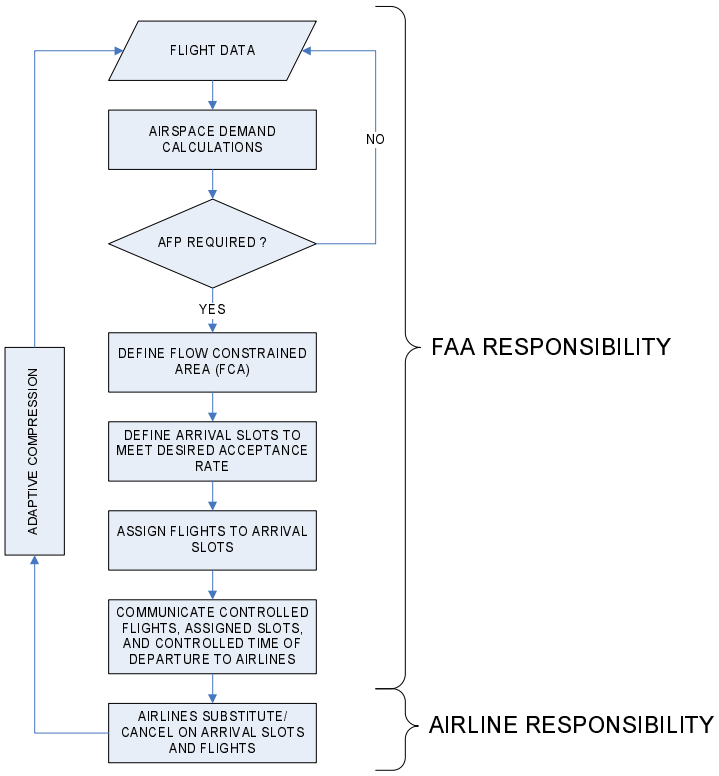

Figure 3: Airspace Flow Program Iterative Process.

2.2 Collaborative Decision-Making and Ground Delay Program Enhancements

Wambsganss (1997) details issues related to the initial implementation of collaborative traffic flow management to Ground Delay Programs (GDP). During reduced arrival capacity at heavily used airports, usually as a result of severe meteorological conditions, the Air Traffic Control System Command Center (ATCSCC) initiates an airspace management procedure known as GDP. The purpose of GDP is to obtain a constant arrival rate consistent with the reduced capacity of the airport. The rationale for this procedure is that flight delays experienced at the origin airport are less costly and safer for the flight operator compared to airborne delays. To accommodate GDPs flights are rescheduled, canceled, or assigned to arrival slots vacated by canceled flights. The noted criticisms of GDPs are that decisions are

based on poor quality data and existing rules (prior to CDM) are inflexible and restrictive.

Wambsganss (1997) discusses how, initially, airlines could be "double penalized" by providing schedule information. Most notable is the restriction that airlines could not substitute flights into arrival slots vacated after the same airline's cancellation of another flight. The concept of "schedule compression" and the compression algorithm are introduced so that airlines are able to substitute into slots vacated by canceled flights in a way that does not conflict with FAA's goal of maximizing system efficiency. The original intent of the compression algorithm was to provide an incentive to participate in CDM to ensure that arrival slots are not underutilized.

System stability and gaming concerns are noted by Wambsganss (1997) as being of prime importance. The data exchange process would be significantly degraded if some NAS operators do not release their information and instead wait for competitor decisions. Strategies and the status of compliance monitoring through the Flight Schedule Monitor (FSM) are outlined. If an air carrier disputes some aspect of a GDP then the reporting features of FSM provide sufficient details to air carriers so that productive discussion and collaboration can be initiated with the FAA and other carriers.

Chang et al. (2001) expands on CDM GDP concepts and implementation details. The structure of messages between air carriers and the FAA through the CDM communications network (CDMnet) is presented. CDM messages inform the FAA, and other CDM participants, of flight cancellations, new flights, and modifications to flight schedules or other operational aspects. Airlines are required to provide an update of each flight within 12 hours of the scheduled departure time. Using this flight information an airport demand list (ADL) is created that includes all airport arrivals during a 16-hour time period. ADL information updates at 5 minute intervals and is distributed to all users requesting the information. A filtering process restricts airlines to accessing information regarding their own flights.

Ball et al. (2003a) synthesized and extended the work of Ball et al. (1998) and Free Flight Phase I Office (1999) in quantifying the impact of CDM on ground delay programs. Several positive impacts were established based on the new information produced by the integration of FAA and airline data sources. After the CDM-enhanced real-time data feed airline operations centers are able to receive the same information as specialists at the ATCSCC. Also,

the real time data available to ATCSCC specialists is of higher quality than the previous data models would allow.

The prediction error for the flight departure time on GDP days is significantly higher than the error experienced when a ground delay program has not been initiated. CDM did not significantly affect the predictive error on days without a ground delay program. But, on days with a GDP the error in departure prediction dropped every month after CDM prototype operations indicating an ongoing improvement in the accuracy of flight departure data.

Airlines are now able, through the CDM network, to directly submit flight cancellation messages in a timely manner. A typical impact of CDM was experienced at San Francisco International Airport (SFO) where the average cancellation notice was at least 29 minutes after the original estimated time of departure without CDM. With CDM operations the cancellation notification time was dramatically improved to the point where with CDM cancellations are received on average 48 minutes before the estimated flight departure. This difference of 77 minutes was noted as conservative by the authors and would most likely be closer to 93 minutes during actual operations.

The better quality information produces a positive impact on user delay. During the 1998 to 1999 study period the delay reduction at major airports ranged from a low of 7.5% at Atlanta Hartsfield Airport to a high of 18.2% savings at Boston Logan Airport. The average for all airports that implemented a GDP was estimated at 12.7%. After CDM the ground delay programs at some airports were also reduced since, in some instances, airlines have prevented capacity-demand imbalances by voluntarily reducing demand in response to CDM information. To accommodate airlines taking corrective action the ATCSCC managers use the improved CDM information to wait until the last possible minute to initiate a GDP.

Ball et al. (2003b) specifies how the Ground-Holding Problem (GHP) can be formulated as a network-flow model variant. The ground-holding problem quantifies the amount of delay imposed on aircraft during a ground delay program. The output of the model is the aggregate number of flights assigned to airport arrival times with no explicit consideration for rationing and equity. This approach, Ball et al. note, can be used in conjunction with a CDM approach that controls arrival times of individual flights.

The modeling idea is to consider arrival flights as the supply (S) nodes and the airport arrival rate (AAR), or airport arrival capacity, as demand (D) nodes. The arrival flights are expressed as deterministic quantities such that the model requires a single estimation of future demand levels. The airport arrival rate has more intrinsic uncertainty and, as such, a probabilistic distribution of AAR is used for analysis. The linear relaxation of the integer program formulation of the GHP presented below is shown to yield integral solutions efficiently.

Minimize

$$\sum_{t=1}^{T}\left[(g)(y_t)+(c)(a_t)+\sum_{q=1}^{Q}(p_q)(a)(z_{tq})\right] \tag{1}$$

subject to:

$$a_t + y_t - y_{t+1} = S_t \qquad \text{for } t \geq 1 \ (y_0 = 0), \tag{2}$$

$$z_{t-1q} + a_t - z_{tq} \leq A_{tq} \qquad \text{for } t \geq 1, q \geq 1 \ (z_{0q=0}), \tag{3}$$

$$a_t, y_t, z_{tq} \geq 0 \tag{4}$$

The decision variables y_t, z_{tq}, and a_t represent the number of flights held on the ground, held in the air, and transitioning from the ground (state 1) to the air (state 2) respectively during the time period t to $t+1$. The costs per time period of holding a flight on the ground, holding a flight in the air, and transitioning between air and ground are g, a, and c respectively. For this analysis the cost of transitioning from state 1 on the ground to state 2 in the air is assumed to be zero. S_t is the number of flights scheduled to arrive during time period t. The AAR during the time period t to $t+1$ under scenario q, with associated probability p_q, is denoted as A_{tq}.

Lulli and Odoni (2007) extends the ground holding problem to consider both airport and en route sector capacities in the context of European air traffic management. Only the time of release of the aircraft into the airspace is considered and airborne delay and re-routing are not modeled. Further simplifications that the authors acknowledge include deterministic demand, deterministic capacity, and equal speed of travel for all flights. The model is applied to study issuies of equity between air carriers.

Carlson (2000) considers the GDP operations problem from an airline's perspective. A model is developed under the CDM framework and explicitly considers operations at hub airports since schedule disruptions are most likely to have the largest impact at these locations. At hub airports several aircraft arrive during brief intervals of time and, after passengers connect to other flights, many aircraft are then scheduled to depart. Only one airport is included in the scope of the model and the rescheduling of outbound flights is not considered. The integer programming model allows airlines to reschedule inbound flights when the airport arrival capacity is reduced with the objective of minimizing the cost of delays and rescheduled flights within the operational constraints and CDM rules. Alternative formulations are developed that include a classical transportation problem representation, however the authors are unable to ensure that the linear relaxation produces the optimal integer solution as in Ball et al. (2003b).

2.3 Free Flight

The Free Flight operational concept allows pilots operating under instrument flight rules (IFR) to select their aircraft's speed, course, and altitude in real time. Air traffic controllers will only intervene if an encroaching aircraft violates the alert zone surrounding another aircraft. This alert zone is larger than the minimum separation shell, or protected zone, that can never be violated. The formal definition developed by the Radio Technical Commission for Aeronautics (RTCA, 1995) is as follows:

> "A safe and efficient flight operating capability under instrument flight rules in which the operators have the freedom to select their path and speed in real time. Air traffic restrictions are only imposed to ensure separation, to preclude exceeding airport capacity, to prevent unauthorized flight through special use airspace, and (otherwise) to ensure safety of flight. Restrictions are limited to extent and duration to correct the identified problem. Any activity which removes restrictions represents a move toward free flight."

Free flight will be a consideration in any air traffic research but its application to this problem may be limited. The complexities caused by weather and the need for a smooth,

orderly flow of traffic through congested airspace generally preclude free flight during capacity disruptions.

2.4 Severe Weather

Severe weather is the predominant cause of aviation accidents (Mahapatra, 1999). In addition to catastrophic crashes, atmospheric turbulence originating from severe weather creates discomfort and injury due to dislocation of cabin objects and passengers. To mitigate against these effects pilots avoid airspace currently experiencing, or those forecasted to experience, severe weather. The avoidance of severe weather by pilots reduces the efficiency and utilization of the airspace. Also, air traffic controllers impose larger than required separations to cope with uncertainties in weather conditions further degrade operational efficiency.

Thunderstorms are the severe weather event of most concern to aviation. Convection, the upward motion of warm air that causes thunderstorms, is most prominent between early March and late October. Though turbulence is the primary concern for pilots, several harmful atmospheric effects may be experienced near a thunderstorm (Table 3, Mahapatra (1999)). Thunderstorms are generally classified into air mass (area) or line (Figure 4). Air mass or single cell random storms have several independent thunderstorms occurring in a less dense system than multi-cell squall lines. Pilots are less likely to pass through line thunderstorms since they have fewer gaps in the system and are generally viewed as more hazardous.

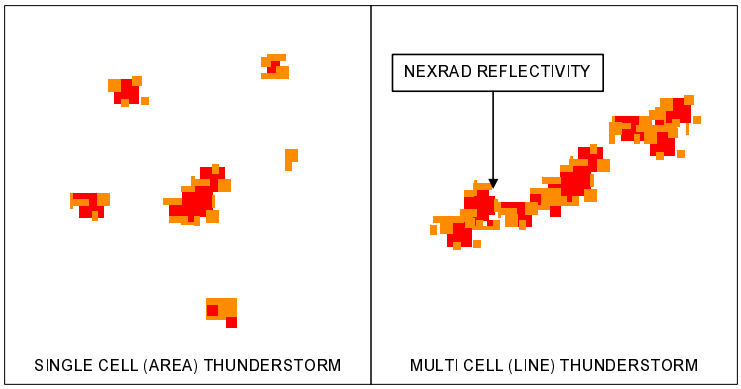

Figure 4: Thunderstorm Classifications.

Table 3: Impact of Thunderstorms on Aviation.

Condition	Description	Impact
Wind Shear	Systematic spatial and/or temporal variation in wind speed	Difficult to control aircraft (especially during landing)
Turbulence	Random motion of air	Pilot loss of control/Increased stresses on aircraft and possible structural failure
Downburst	Burst of outward winds at low altitudes	Loss of control
Rain	Liquid water in atmosphere	Erratic engine combustion and possible loss of engine power ("flameout")
Hail	Spherical or irregular pellets of ice	Possible structural damage and/or damage to interior parts during engine ingestion
Icing	Ice formation on aircraft at altitudes above "freezing height"	Loss of lift and increased drag
Electrical phenomenon	Natural lightning or lightning triggered by aircraft passing through electric field potential gradients	Generally benign for aircraft with electrically conductive skins (e.g. aluminum)

The primary aviation weather sensor in use today is the Next Generation Weather Radar (NEXRAD WSR-88D). The NEXRAD systems functions by beaming electromagnetic energy into an observation space and detecting objects, most commonly rain and hail, by sensing the energy reflected. The velocity of airborne objects may also be estimated by exploiting the Doppler Effect. Detection of thunderstorms using NEXRAD is accomplished by comparing the reflectivity field of the radar data, height of the atmospheric event, and weather spatial extent to known thunderstorm characteristics. The NEXRAD system currently operates at an approximate 90% detection rate with 20% false alarms (Mahapatra, 1999). Early detection is based on monitoring the precipitation activity at high altitudes by varying some of the NEXRAD operational parameters and characteristics.

Airspace users rely on National Oceanic and Atmospheric Administration (NOAA) avi-

ation weather products (http://adds.aviationweather.noaa.gov/) to decide if future aircraft departures will encounter hazardous weather en route, and, if significant gaps will exist to route aircraft through the evolving weather system. The Collaborative Convective Forecast Product (CCFP) provides three maps of 2-, 4-, and 6-hour forecasts updated every two hours. Convective areas are depicted as polygons with coverage, type (line or area), tops, growth rate, movement, and the forecaster's subjective confidence that the event will occur. Due to the uncertainty in current and forecasted severe weather effects several participants collaborate to create the forecast. A portion of a historical collaboration log is shown in Fiure 5.

The National Convective Weather Forecast (NCWF) complements CCFP by providing the current +1 hour forecast of convective weather. NCWF is generally used by the aviation community to update severe weather avoidance strategies based on initial data from the CCFP. ARTCC traffic managers view the 1-2 hour forecast at the 100-200 nautical mile spatial extent as most desirable for strategic decision-making (Forman et al., 1999).

NOAA and NCAR (National Center for Atmospheric Research) have collaborated since 1997 to develop an aviation weather forecast verification system (Mahoney et al., 2002). The system, referred to as the Real Time Verification System (RTVS), seeks to provide consistent and unbiased evaluations of CCFP, NCWF, and other operational or experimental aviation weather products.

The RTVS methodology is as follows. The forecast region is subdivided into an observation grid of 4, 20, or 40 km depending on the forecast type. The user has the ability to specify different regions (National, West, Central, East, or Northeast Corridor) since the northeastern region of the United States has better radar coverage and will produce superior forecasts when compared to western regions. RTVS then performs a pair wise comparison of forecast and observation for each grid (Table 4). The convention used throughout RTVS is to refer to the first digit as the forecast (Y = forecasted convection; N = no convection forecasted) and the corresponding second digit as the observation (Y = convection observed; N = no convection detected). Based on this paired system several appropriate statistics such as false alarm rate and probability of detection are evaluated (Table 5).

A web front end for RTVS (http://www-ad.fsl.noaa.gov/afra/rtvs) can be used to access

```
COLLABORATION PARTICIPANTS:
| ATCSCC | ATCSCCSP | AWC-Forecaster | EJA | MSC-Forecaster | NWA | UAL
| ZAB | ZDC | ZFW | ZHU | ZID | ZJX | ZKC | ZME | ZMP | ZNY | ZOB |
18 Participants Logged into Collaboration
9 Participants Contributing to Collaboration
******************************
AWC-Forecaster 12:14:42Z
Good morning...
AWC-Forecaster 12:15:38Z
Continued with forecast trends and added three areas for 19z...
AWC-Forecaster 12:17:27Z
Area over northern Plains is consolidating in region of good lift/instability
 and will continue into northern/central MN.
AWC-Forecaster 12:18:50Z
Areas in TX is having difficulty in developing assoc with shortwave moving
 east but TCU are on in the increase along the coastline so will continue with
 forecast trend there.
AWC-Forecaster 12:20:24Z
Area ts over AR/LA has decreased some recently but still appears to be in a
 favorable location given apparent vort location over central AR and instability
 axis over ne LA/se AR so will continue with area there.
AWC-Forecaster 12:21:32Z
Developed new area over LA/MS at 19z as models indicate increasing lift/conv
 qpf about then.
ZHU 12:23:22Z
TX looks pretty goodas ts shd increase next couple of hours. May want to expand
 srn edge back towards RIO Grande at 15Z.
ZMP 12:23:33Z
maps look good for ZMP.
AWC-Forecaster 12:23:58Z
Pockets of higher instability/surface covergence/slight lift may be enough to
 initiate small areas of ts near the stationary/cold front IL to NY by 19z.
```

Figure 5: Sample Collaborative Convection Forecast Log.

Table 4: Forecast Observation Comparison Convention is RTVS.

Forecast	Observation		Total
	Yes	No	
Yes	YY	YN	YY+YN
No	NY	NN	NY+NN
Total	YY+NY	YN+NN	YY+YN+NY+NN

Table 5: RTVS Output Statistics.

Statistic	Formula	Description
Probability of Yes Detection	$POD_y = \frac{YY}{YY+NY}$	Proportion of Yes observations forecasted
Probability of No Detection	$POD_y = \frac{YY}{YY+NY}$	Proportion of No observations forecasted
False Alarm Ratio	$FAR = \frac{YN}{YY+YN}$	Proportion of incorrect Yes forecasts
Critical Success Index	$CSI = \frac{YY}{YY+YN+NY}$	Ratio of correct Yes forecasts to Yes forecasts or observations
True Skill Statistic	$TSS = POD_y + POD_n 1$	Discrimination measure
Proportion Correct	$PC = \frac{YY+NN}{YY+NY+YN+NN}$	Proportion of forecasts matching observations
Bias	$Bias = \frac{YY+YN}{YY+NY}$	Yes forecasts relative to Yes observations

these statistics. Examples presented in Figure 6 shows a time series analysis of the NCWF 1 hour forecast. Based on similar data for different time periods researchers have concluded that forecasters routinely predict more severe convective weather conditions than are likely to occur (Kay et al., 2006).

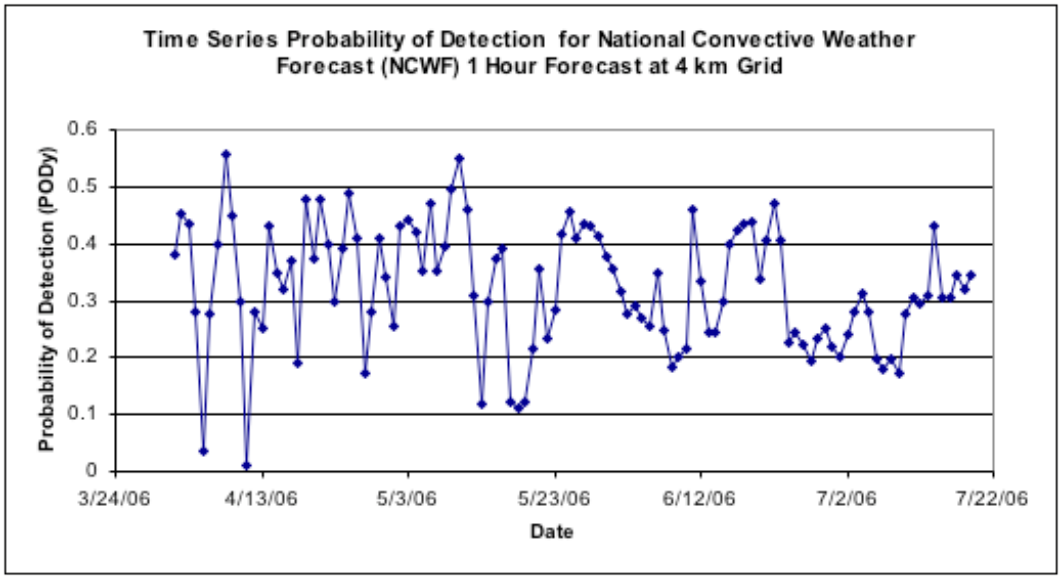

Figure 6: Sample RTVS Time Series Output for NCWF.

Rhoda et al. (2002) studies the deviations of pilots from nominal flight routes due to weather. The human analysts studied 43.5 hours of flight track data during six thunderstorms in 1999 over the Memphis, Tennessee Center. Rhoda et al. note that the main difficulty with this study is the identification of deviations since en route aircraft can generally avoid the storm edge by adjusting their course by as little as 10 to 20 degrees. The results indicate that pilots almost never penetrate level 2+ (moderate) precipitation in the en route regime. Conversely, pilots routinely penetrate level 2+ or greater precipitation in the terminal approach since the cost of deviating in the terminal area is higher than en route. Terminal area deviations require an aircraft to forfeit position in the landing queue and are subsequently moved to the end of the queue. The prediction model generalized that pilots deviate when echo tops are near or above them although pilots do not avoid flying over thunderstorms.

Evans et al. (2004) attempts to quantify the delay impact of convective weather. Evans

et al. note that convective weather will reduce the capacity of an airport but rarely will thunderstorms close traffic to both arrivals and departures for more than 15 minutes. Other than a reduction in visibility, the major impact of thunderstorms is the potential for closing arrival and or departure fixes. An arrival fix is a location in three-dimensional space that aircraft use to create a more organized landing pattern for controllers. Similarly, a departure fix is for aircraft taking off from a airport runway. Fixes may be altered depending on the runway configuration used at the airport. If convective weather effectively closes the arrival fix then arrivals may be routed through the departure fix and departures halted.

Another significant impact of severe weather is the "delay ripple effect" or the need for "delay multipliers". An aircraft may be used for multiple flight legs in a single day and if adverse weather impacts one leg then subsequent legs will experience increasing levels of gate departure delays. A study by American Airlines and Oak Ridge National Laboratory estimated delay multipliers as high as four for early hours in the day and a more typical value of 1.8 in the mid afternoon (Beatty et al., 1999).

2.5 Stochastic Elements of National Airspace System

There is a body of work in the recent literature focusing on quantifying and modeling stochastic elements of the NAS en route airspace. The estimated time of departure is the single largest source of uncertainty for flights that have not departed from the origin airport (Wanke et al., 2004). The work on pre-departure uncertainty has focused on quantifying variance and confidence bounds under various weather and flight-specific attributes at a range of look-ahead times (Lindsay et al., 2005; Krozel et al., 2002; Mohleji and Tene, 2006).

The prediction of departure time is one component in the estimation of en route airspace sector demand. Meyn details a method to estimate sector and airport demand from arrival probability distributions and sector traversal time (Meyn, 2002). Only a single source of uncertainty is modeled at an unspecified look-ahead time. Mueller et al. note that departure time, wind forecasting errors, deviations from the flight plan, and aircraft performance and weight uncertainty can lead to errors in sector demand prediction (Mueller et al., 2002). The climb phase of flight, especially step climbs that are mandated by air traffic controllers in congested airspace, is identified as the source of the highest trajectory performance prediction

errors with empirical results presented.

Flow models are another proposed approach to improve the estimation of sector demand by considering air traffic demand at a high level. Many of the current models are deterministic though well-suited to metering traffic flows to an arrival fix at a busy airport (Menon et al., 2004; Bayen et al., 2005; Dugail et al., 2002; Menon et al., 2006). Probabilistic versions have also been developed but at the more macroscopic center level (Sridhar et al., 2006). An attempt to establish a relationship between planned and observed sector counts is discussed in Gwiggner and Duong (2006).

Krozel et al. (2002) used air traffic controller interviews, NAS aggregate conditions, and center data to classify the impact of various sources on en route sector demand. Krozel et al. concluded that traffic flow management (TFM) initiatives, air traffic control actions, and take-off time are the most influential sources of en route sector demand error (Table 6). Another important result is that filed flight plans are typically inaccurate by several minutes by the time the aircraft enters the sector. It is of prime importance to include other sources of data to accurately predict sector arrivals for strategic decision-making. Recommendations include producing additional electronic data relating to holding patterns, ground stops, de-icing conditions, etc. so that more accurate trajectory predictions can be made.

Lindsay et al. (2005) attempts to quantify the error in departure time and predicted sector entry time for aircraft operating in good weather and bad weather days. Lindsay et al. emphasize that the primary source of prediction error for en route sectors is the departure time of aircraft and not the en route traversal time. The data for several scenarios are presented that indicate a prediction error of approximately 4 to 10 minutes at takeoff, 7 to 10 minutes at 1 hour prior to departure, and 16 to 21 minutes at 2 hours prior to departure. Error is defined as the absolute difference between scheduled and actual departure times.

2.6 Conflict Detection and Resolution

FAA (2006b) describes required air traffic control procedures including minimum separation standards between aircraft. In the en route airspace aircraft must be separated at least 1000 feet vertically or at least 3 nautical miles horizontally. The minimum horizontal separation increases to 5 nautical miles for aircraft greater than 40 miles from the radar site. When

Table 6: Sources of En Route Demand Prediction Error. Influence Defined in Krozel et al. (2002).

Influence	Source
High	Departure Time Prediction
	runway/taxiway closures
	obstructions on runway/taxiway
	snow/ice removal
	de-icing operations
	runway direction reversals
	unavailable gates
	general aviation aircraft without flight plans
	accidents/incidents
	TFM Restrictions and ATC Actions
	non-standard procedures between sectors/centers
	controller style/preference
	insufficient data on TFM initiatives
	daily and seasonal trends
	Horizontal Route Prediction Accuracy
Medium	Vertical Route Prediction Accuracy
	Flight Speed Prediction Accuracy
	Changing Airspace Adaption Data
	Weather and Winds Aloft Forecast Accuracy
Low	Surveillance Data
	Flight Technical and Operational Errors
	Trajectory Models

separating aircraft vertically controllers must assign new altitudes based on course. If the aircraft course is between 0 and 179 degrees from magnetic north an odd cardinal altitude must be assigned (e.g. 31,000 feet), otherwise an even cardinal altitude is used (e.g. 34,000 feet). Bayen et al. (2005) describes common maneuvers air traffic controllers issue to pilots to prevent violations of minimum separation standards, otherwise known as conflicts (Figure 7).

Generally holding patterns are the least preferred alternative since they require constant monitoring by controllers.

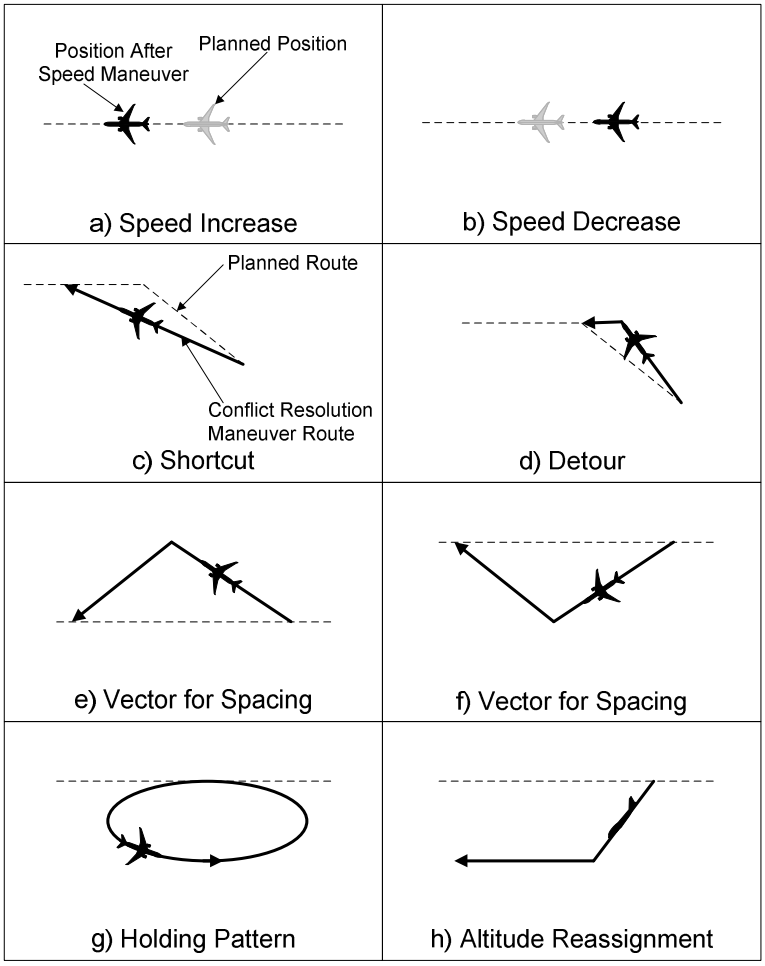

Figure 7: Conflict Resolution Maneuvers.

Kuchar and Yang (2000) review several methods and decision support systems for aircraft conflict detection, alerting, and resolution. Conflicts refer to violations, or impending violations, of minimum airborne separation between aircraft. Kuchar and Yang provide an overview of the conflict detection and resolution process, classifies the modeling approaches, and describe system design and implementation issues.

All of the systems require some form of a dynamic trajectory prediction model. The trajectory model is used to establish metrics such as estimated time to loss of minimum separation. Based on these metrics, and a specified operator alert threshold, a decision is

made whether to inform the human users.

Kuchar and Yang state that the most significant difference between models is the trajectory extrapolation method (Figure 8). The traffic alert and collision avoidance system (TCAS) that has been on-board domestic aircraft since the early 1990's uses a nominal trajectory prediction. Nominal predictions, or expected values, are most useful for short term horizons and near-miss scenarios.

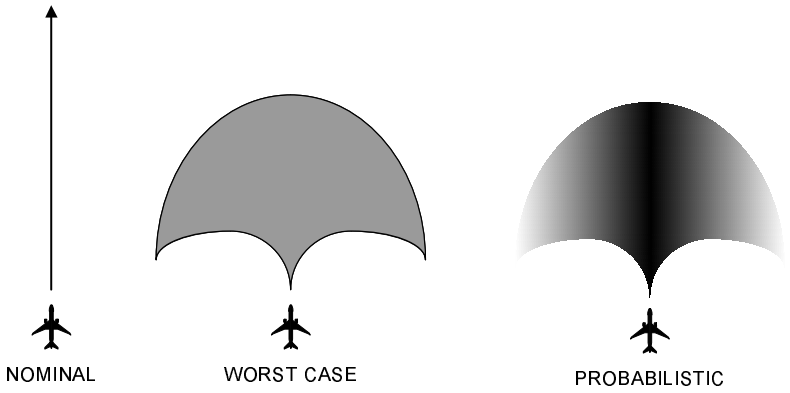

Figure 8: Trajectory Prediction for Conflict Detection.

A worst case analysis considers all possible trajectories to determine the potential for conflicts. Worst case is useful for predicting an upper bound on the number of conflicts or deciding if a conflict has been resolved. The probabilistic approach uses potential uncertainties in future aircraft position to describe a range of possible scenarios. The advantage of a probabilistic approach is that decisions can be made based on the likelihood of a conflict.

The conflict resolution solution may be categorized into prescribed, optimized, force field, or manual. Prescribed conflict resolution is based on a set of predefined procedures such as a fixed climbing turn maneuver. The perceived benefit of prescribed procedures is that operators can be trained to perform them instinctively. However, manual resolution procedures cannot respond to unexpected events and are therefore less effective in general.

An optimal resolution approach calculates trajectories with the lowest estimated cost. The cost, or objective, is a combination of simple economic values and considerably more complex human workload and utility indicators. Researchers have used techniques such as game theory, genetic algorithms, expert systems, and fuzzy control to obtain a solution.

By treating each aircraft as a charged particle and using modified electrostatic equations a force field technique is defined to generate resolution maneuvers. Kuchar and Yang identify potential problems such as the requirement for continuous adjustment to speed and altitude with the potential for sharp changes (discontinuities) in resolution maneuvers.

The manual system depends on the user to generate trial solutions that are then evaluated. The noted advantage to this system is the capability of human intuition to solve problems.

Kuchar and Yang observe that no single system has been clearly shown to be the most effective or efficient. Also, Kuchar and Yang caution that models have differing levels of validation and those perceived as less complex but are used in practice may be more reliable than complex research models.

2.7 National Airspace System (NAS)

The NAS includes airspace over the US from 60,000 feet down to, but excluding, the ground (FAA, 2000). Airspace is divided into five controlled categories, airspace Classes A through E, and the uncontrolled Class G airspace. These categories are established based on the complexity and/or density of aircraft movements, the nature of operations in the airspace class, safety requirements, and the public interest. Class A is defined by more restrictive regulatory controls than Class B, Class B more restrictive than Class C and so on. En route commercial aircraft typically operate in Class A airspace at an altitude between 18,000 feet Mean Sea Level (MSL) and 60,000 feet MSL. The airspace up to 10,000 feet MSL surrounding the busiest commercial airports defines Class B airspace. Further information regarding these and other airspace classifications can be found in the Aviation Capacity Enhancement Plan (FAA and ARP Consulting, 2000).

Special Use Airspace (SUA) further restricts commercial air traffic from prohibited, restricted, military operations, alert, warning, and controlled firing areas. Flights over sensitive facilities such as the White House are prohibited without authorization from the FAA and other agencies. Restricted airspace contains atypical hazards, such as missile testing, and differs from prohibited airspace by allowing some aircraft to operate during specific hours. Restricted areas over domestic and international waters are referred to as warning areas.

Military operations areas are established to contain military activities though flights remain permissible in this airspace.

FAA air traffic control (ATC) services maintain minimum separation standards between aircraft and provide navigational assistance to pilots. The safe operations of aircraft on the ground and up to 5 nautical miles from airports falls within the domain of air traffic control towers (ATCTs). Terminal approach control facilities (TRACONs) direct traffic on final approach and departing from an airport facility. Air traffic en route is controlled by twenty air route traffic control centers (ARTCCs). Each of these ARTCCs is further divided into approximately 20 to 50 sectors.

2.8 Airspace Sector Occupancy Model (AOM)

A flight will traverse one or more airspace sectors during its journey and may impact sector operations and trajectories of other flights. Sherali et al. (2000) mathematically describe the sequence of sectors that a flight will occupy in an airspace sector occupancy model (AOM). The geometric description of sector modules and flight trajectories are the primary inputs to AOM.

Flight paths are described by a series of waypoints that break the trajectory into linear segments. Starting with a waypoint of known sector occupancy, generally the origin airport, the trajectory is traced until a ceiling, floor, or vertical face is encountered. The trajectory exits the module through the face where $F_{ps} \cdot d < 0$ where F_{ps} is the inward gradient orthogonal to face p in sector s. d is the direction from the current waypoint toward the next waypoint. If the next waypoint in the flight trajectory is encountered before the sector boundary then the flight is advanced to the next waypoint and a new direction, d, is established.

Once the intersection between the flight trajectory and a sector face is established adjacency information is used to determine possible sectors that the flight will enter. The process is then repeated until the last waypoint is encountered. The work of Sherali et al. (2000) is broader in scope than AOM, but the airspace planning model described therein is developed into more robust models by Sherali et al. (2002) and Sherali et al. (2003).

2.9 Airspace Planning and Collaborative Decision-Making Model (APCDM)

Sherali et al. (2003, 2006) describes an airspace planning and collaborative decision-making model (APCDM) with the goal of effective management of air traffic. The management of airspace is most challenging during airspace disruptions such as severe weather and other restrictions. The APCDM model includes the AOM model of Sherali et al. (2000) and extends preliminary concepts described in Sherali et al. (2002).

The probabilistic aircraft encounter model (PAEM) examines aircraft trajectory pairs and estimates the probability of a Level-1, Level-2, or fatal conflict occurring. A Level-1 conflict occurs when an intruder aircraft enters the standard separation shell of a focal aircraft. Level-2 and fatal conflicts are similarly defined with a half-dimension separation shell and a tight enveloping shell of dimension (500 feet in-trail, 500 feet cross-track, 100 feet altitude) respectively. The probabilistic aspect of PAEM considers errors in aircraft position and creates additional realizations of the aircraft nominal trajectory for consideration in conflict analysis. Also notable is the fact that the AOM and PAEM models are preprocessed as inputs into the optimization scheme.

Air traffic controllers must be able to safely resolve conflicts predicted in the PAEM. To this purpose the workload constraint generation restricts the number of simultaneous Level-1 and Level-2 conflicts in a sector of interest. Fatal conflicts are prevented from occurring at all. Another constraint restricts the maximum simultaneous number of aircraft occupying a sector.

The concepts of "airline collaboration cost" and "airline collaboration efficiency" are introduced as part of the representation of equity in a CDM framework. These are used to indicate how costly a proposed solution is relative to a specific airline's least cost solution. If the normalized cost indicator, collaboration efficiency, for each of the airlines is comparable then collaboration equity is approached. The model formulation, after sector occupancy and conflict analysis preprocessing, is shown below.

Minimize

$$\sum_{f=1}^{F}\sum_{p\in P_{f0}} c_{fp}x_{fp} + \sum_{s=1}^{S}\sum_{n=0}^{\bar{n}_s}\mu_{sn}y_{sn} + \mu^D\sum_{\alpha=1}^{\bar{\alpha}}\omega_\alpha[1-E_\alpha(x)] + \mu^e x^e + \mu^e_{max}E^e_{max} \tag{5}$$
$$+\sum_{s=1}^{S}\gamma_s w_s + \sum_{(P,Q)\in A}\phi_{PQ}z_{PQ}$$

subject to:

$$\sum_{p\in P_{f0}} x_{fp} = 1, \qquad \forall f = 1,\ldots,F \tag{6}$$

$$\sum_{(f,p)\in C_{si}} x_{fp} \le n_s, \qquad \forall i = 1,\ldots,\ I_s, s = 1,\ldots,S \tag{7}$$

$$w_s = \frac{1}{H}\sum_{(f,p)\in\Omega_s} t^s_{fp}x_{fp}, \qquad \forall s = 1,\ldots,S \tag{8}$$

$$n_s - w_s = \sum_{n=0}^{\bar{n}_s} n y_{sn}, \qquad \forall s = 1,\ldots,S \tag{9}$$

$$\sum_{n=0}^{\bar{n}_s} y_{sn} = 1, \qquad \forall s = 1,\ldots,S \tag{10}$$

$$x_P + x_Q \le 1, \qquad \forall (P,Q)\in FC \tag{11}$$

$$\sum_{(P,Q)\in M_{sk}} z_{PQ} \le r_s, \qquad \forall k = 1,\ldots,K_s,\ s = 1,\ldots,S \tag{12}$$

$$x_P + x_Q - z_{PQ} \le 1, \qquad \forall (P,Q)\in A \tag{13}$$

$$\sum_{Q\in J_{sk}(P)} z_{PQ} \le r_s x_p, \qquad \forall P\in N_{sk} : |J_{sk}(P)| \ge r_s + 1, \tag{14}$$
$$\forall k = 1,\ldots,K,\ s = 1,\ldots,S$$

$$E_\alpha(x) = \frac{D_{max}\sum\limits_{f\in A_\alpha} W_f c^*_f - \sum\limits_{f\in A_\alpha}\sum\limits_{p\in P_{f0}} W_f c_{fp}x_{fp}}{(D_{max}-1)\sum\limits_{f\in A_\alpha} W_f c^*_f}, \qquad \forall\alpha = 1,\ldots,\bar{\alpha} \tag{15}$$

$$E^e_\alpha(x) = E_\alpha(x) - \left(\sum_{\alpha=1}^{\bar{\alpha}}\omega_\alpha E_\alpha(x)\right), \qquad \forall\alpha = 1,\ldots,\bar{\alpha} \tag{16}$$

$$\nu_\alpha \geq -E^e_\alpha(x),\ \nu_\alpha \geq E^e_\alpha(x), \qquad \forall \alpha = 1, \ldots, \bar{\alpha} \tag{17}$$

$$x^e = \sum_{\alpha=1}^{\bar{\alpha}} \omega_\alpha \nu_\alpha \tag{18}$$

$$z_{PQ} \geq 0, \qquad \forall (P,Q) \in A,\ x_{fp} \text{ binary},\ \forall p \in P_{f0},\ \forall f = 1, \ldots, F, \tag{19}$$

$$y_{sn} \geq 0, \qquad \forall n = 1, \ldots, \bar{n}_s,\ s = 1, \ldots, S, \tag{20}$$

$$E_\alpha(x) \geq 0, \qquad \forall \alpha = 1, \ldots, \bar{\alpha}, \tag{21}$$

$$n_s \leq \bar{n}_s, \qquad \forall s = 1, \ldots, S, \tag{22}$$

$$x^e \leq \nu^e, E^e_{max} \geq -\omega_\alpha E^e_\alpha(x) \qquad \forall \alpha = 1, \ldots, \bar{\alpha} \tag{23}$$

Significant indicators of degraded sector performance are included in the objective function (Equation 5). The total cost of selecting a set of flight plans, $\sum_{f=1}^{F} \sum_{p \in P_{f0}} c_{fp} x_{fp}$, is represented as the product of the cost of flight plan p for flight f, c_{fp}, and the binary decision variable for selecting flights, x_{fp}. The components of c_{fp} are fuel and delay costs. The set of alternative flight plans, P_{f0}, includes all surrogates and the flight cancellation event.

The penalty term $\sum_{s=1}^{S} \sum_{n=0}^{\bar{n}_s} \mu_{sn} y_{sn}$ has the effect of minimizing variability of the peak monitoring workload from the average workload of a sector s. μ_{sn} is a model parameter that degrades the solution for situations with sector workloads that significantly exceed the average sector workload during peak monitoring conditions. The y-variables, y_{sn}, represent convex combination weights as defined in Equations 10 and 20.

The average occupancy workload ($w_s = \frac{1}{H} \sum_{(f,p) \in \Omega_s} t^s_{fp} x_{fp}$) of Equation 8 is a summation of the total occupancy time for a flight plan p of flight f in sector s, denoted t^s_{fp}, for selected flights ($x_{fp} = 1$) during a time horizon H. The average occupancy workload is penalized in the objective function in a linear fashion, $\sum_{s=1}^{S} \gamma_s w_s$, where γ_s is a penalty factor. The variability between the peak monitoring workload, n_s, and the average workload is $n_s - w_s = \sum_{n=0}^{\bar{n}_s} n y_{sn}$ with n and y_{sn} defined as previously.

Equity and efficiency in airline operations under the CDM philosophy is represented by $\mu^D \sum_{\alpha=1}^{\bar{\alpha}} \omega_\alpha [1 - E_\alpha(x)] + \mu^e x^e + \mu^e_{max} E^e_{max}$ in the objective function. The airline collaboration efficiency, $E_\alpha(x)$, is a linear function normalized between 0 and 1 as calculated in Equation 15. $E_\alpha(x) = 1$ represents the case where airline α attains its least cost solution. Conversely, $E_\alpha(x) = 0$ is the most costly case allowed for airline α. The weighting factor, ω_α, determines how much an effect $E_\alpha(x)$ for airline α has on the inefficiency term $\mu^D \sum_{\alpha=1}^{\bar{\alpha}} \omega_\alpha [1 - E_\alpha(x)]$

in the objective function. The authors propose the selection of ω_α based on proportion of flights operated by an airline or the proportion of passengers/passenger-miles serviced by an airline α. μ^D in the first equity term, and μ^e, μ^e_{max} in the second and third equity terms, are nonnegative parameters. The second collaboration efficiency term, $\mu^e x^e$, attempts to minimize the variance of airline collaboration efficiencies from the w-mean collaboration efficiency and x^e calculated using Equations 17 and 18. E^e_{max} (Equation 23) is another measure of dispersion for the weighted equity values with higher ranges degrading the solution ($\mu^e_{max} E^e_{max}$).

The conflict resolution workload formulation, represented as a conflict Gantt chart during constraint generation, is summarized by Equations 11-14 and 19. Fatal conflicts are prohibited by $x_P + x_Q \leq 1$ for all flight plan pairs (P, Q) that could potentially result in a fatal conflict. Restricting the simultaneous number of non-fatal conflicts in sector s to be no more than a designated workload parameter r_s establishes the conflict resolution workload constraint $\sum_{(P,Q)\in M_{sk}} z_{PQ} \leq r_s$ where $x_P + x_Q - z_{PQ} \leq 1$ and the objective function penalty term $\sum_{(P,Q)\in A} \phi_{PQ} z_{PQ}$ defines z_{PQ} to be 1 if both binary decision variables x_P and x_Q equal 1. ϕ_{PQ} is a parameter to be estimated based on the conflict characteristics. Equation 14 was developed as a formulation that further restricts the number of simultaneous conflicts in a more computationally efficient manner than that described in Sherali et al. (2002).

The main criticism of APCDM is that the model has not been sufficiently validated using historical data. Also, the model may not be ready for real-time implementation due to computational complexity and processing issues. Several other practical issues must also be considered when comparing the proposed system to the National Severe Weather Playbook, the current benchmark.

The Playbook employs large-scale macroscopic routing strategies during airspace restriction scenarios. This macroscopic strategy allows all involved to receive feedback regarding the intentions of other airspace users. Conversely APCDM performs analysis at the level of individual aircraft trajectories. If airspace users are unable to ascertain, on a macroscopic scale, the intentions of other users then good routing decisions are not possible. Without appropriate alternative trajectories APCDM is a less effective model.

An "all-at-once" optimization strategy in APCDM makes decisions for all flights simulta-

neously and disregards differences in uncertainty between aircraft. There is less uncertainty in 4-dimensional position for an airborne aircraft expected to enter a sector in the next 5 minutes compared to a flight scheduled to depart in the next hour but APCDM treats the uncertainty equally for these flights. It can be argued that the Playbook uses an all-at-once strategy but the Playbook system is more flexible and better specified in terms of which aircraft will be affected and the time frame to implement the traffic flow management initiative.

The Playbook disregards potential violations of minimum aircraft separation standards, also known as conflicts. APCDM explicitly restricts the expected number of conflicts per sector, an improvement over the Playbook. However, some aspects of conflict modeling are questionable. APCDM defines a separate class of conflicts known as "fatal" conflicts corresponding to a smaller separation shell one-half the dimensions of the minimum requirement. Fatal conflicts are more restricted by APCDM contrary to current human factors research that indicates fatal conflicts are detected earlier and require less drastic conflict resolution actions (Wickens et al., 1997). Also, while APCDM was developed to run approximately one hour in advance of an anticipated airspace restriction it is generally acknowledged that conflicts between two aircraft cannot be predicted more than 10 to 15 minutes in advance (Wanke et al., 2005).

Sector capacities, along with conflicts, are the common metrics associated with air traffic controller workload. APCDM improves on the Playbook by modeling sector capacity but key aspects are not considered. Controller performance though practically considered a deterministic quantity in APCDM is always stochastic, even for the same controller. Traffic complexity, sector configuration, proximity to the weather event, and a myriad of other factors can impact controller performance (Wickens et al., 1997). Furthermore, even though sector capacity is modeled as a "hard" constraint the capability should exist to examine the benefits of temporarily violating capacity and metering traffic at the sector boundary.

Wanke et al. (2005) suggests an alternative to APCDM based on the probabilistic modeling of severe weather, demand, and sector capacity (controller workload). The focus of the research is primarily related to quantifying sources of uncertainty in the NAS and presenting this information appropriately to traffic flow managers. The impact of severe weather is

limited to the probability of blocking flow paths through a sector. To deal with uncertainties in demand, traffic is segregated into active and proposed flights and levels of uncertainties are assigned to each of these stages of flight. Lastly, the authors propose relating traffic flow patterns to sector capacity in a task-based approach. Wanke et al. note a current research direction of using heuristics to decide when to implement a TFM initiative and which TFM initiative to use for a particular situation.

Wanke and Greenbaum (2007) extend earlier work by introducing a decision tree approach to deal with system uncertainty. The approach details a method in which decisions are delayed until better information is received. At each decision point a series of Monte Carlo simulations are executed in parallel to estimate the system wide cost of each decision. Simulation of the decision tree approach shows promising results but explicit details of the computational methods are not presented.

Bertsimas and Patterson (1998) formulates the en route air traffic management problem as a dynamic multicommodity network flow problem (Bertsimas and Patterson, 1998, 2000). The model was the first to control the departure time (ground holding time), route through the NAS, and speed. The authors note that the linear solution to the integer program often results in a feasible integer solution, a desirable property. In cases where an integer solution is not obtained a set of heuristics and routing schemes is presented to achieve integrality. Unfortunately equity between airlines is discussed but not implemented in the constraint list. Simplifying assumptions include deterministic demand, deterministic capacity, and all aircraft are assumed to have the same performance regardless of performance or wind conditions.

An interesting proposal for the management of en route airspace in Europe is to specify direct routes for all flights and separate them vertically by assigning unique flight levels for conflicting flights. Barnier and Brisset (2004) derives a method using graph coloring heuristics to solve a simplified version of the problem. However, this approach does not correspond with current FAA practice and during severe weather there is no resolution mechanism for flights that cannot climb above the weather system.

A model that focuses on limiting airspace complexity (Section 2.10) in Europe is presented by Flener et al. (2007). A heuristic solution method is formulated that considers changes to

departure time, en route airspeed, or altitude. This method models important considerations but has limited applicability for this problem due to differences in European airspace design and the lack of consideration for severe weather.

2.10 Human Factors and Sector Capacity

The safe and efficient operation of the NAS is ultimately the responsibility of human operators. Unfortunately air traffic controllers are subject to errors and performance breakdowns that can compromise goals of safety and efficiency. There are numerous causal factors for poor operator performance but the discussion here will be limited to workload as it relates to airspace performance. Workload refers to the mental effort, or subjectively perceived mental effort, exerted by a human operator in response to a task. Workload can also be defined in terms of the task load demands and performance response of the operator (Wickens et al., 1997).

The majority of research in this area has focused on controller response to high levels of workload. Attentional resource theories (Wickens, 1992) predict a performance decrease with increasing levels of task load. Endsley and Rodgers (1996) found a positive correlation between workload and operational errors for conditions that present operators with the highest task demands. This result supports the theoretical prediction that higher levels of workload will have a negative impact on safety and traffic-handling capacity. However not all conditions lead to this result and it is generally accepted that the relationship between mental workload and performance is not straightforward. Experienced controllers are able to use adaptive strategies to cope with short-term increases in traffic density and complexity. Other less important tasks can be suspended or transferred to a colleague. Controllers may also increase aircraft spacing or prevent aircraft from entering a sector to increase decision time and decrease complexity. Some studies suggest that it is possible for controllers to maintain performance under increased workload (Wickens et al., 1997).

The FAA has introduced automation in an attempt to reduce the operator's workload during situations of high task load. Automation can potentially reduce workload depending on the task but in practice other unintended effects may counteract automation benefits. There is some evidence that automating certain tasks actually increases workload (Wiener,

1988). In general the effects of automation have been mixed and substantial operator performance increases have yet to be observed.

Several workload drivers, or factors than increase mental effort, have been proposed. Arad (1964) presents some factors that are typically considered as workload drivers (Table 7). Several other factors may also be used as indicators of task load imposed on the air traffic controller but since there is considerable diversity in response to workload among these controllers the process of assigning relative weights is challenging.

Table 7: Air Traffic Control Load Variables (Arad, 1964).

Number of controlled aircraft
Sector flight time
Proportion of standard/non-standard aircraft
Proportion of terminal area hand-offs
Proportion transitioning
Proportion of VFR to IFR "pop-ups"
Mean airspeed
Sector area
Mean aircraft separation

Operational errors are also reported during low to moderate traffic complexity conditions. During low task load controllers may experience boredom and reduced alertness, which has serious implications for emergency situations. Rodgers (1993) and the Canadian Aviation Safety Board (1990) have concluded that most operational errors occur during low traffic complexity situations.

Majumdar et al. (2005) reviews the application of basic human factors research to the estimation of sector capacity used in a traffic flow management context. Majumdar et al. describe the process of estimating sector capacity using fast-time simulators (computer software) and real-time simulators (human-in-the-loop studies). Real-time simulators are required since the amount of air traffic alone cannot fully explain a controller's workload (Mogford et al., 1995). Two differing methods to estimate workload are presented. The first is based on developing a formula to "score" workload based on several predictors such as

simultaneous aircraft occupying a sector and sector crossing time. Based on the real-time simulator a threshold score is developed to define capacity. The second method is a task-based approach whereby each monitoring task is assigned a specified time for a controller to complete. Capacity for task-based methods is commonly based on the number of minutes in one hour used for monitoring tasks, e.g. 42 minutes in one hour (70%).

The FAA uses facility voice tapes to develop a task-based workload estimation for sector air traffic controllers. Rodgers and Drechsler (1993) classifies controller activities (Table 8) considering the tactical (R-side) and planning (D-side) sector controllers as a team . Leiden and Green (2000) differentiates the responsibilities (Table 9) of the tactical and planning controllers based on FAA regulations (FAA, 2006b).

Table 8: Top-Level Controller Activities (Rodgers and Drechsler, 1993).

1)	Perform Situation Monitoring
2)	Resolve Aircraft Conflicts
3)	Manage Air Traffic Sequences
4)	Plan Flights
5)	Assess Weather Impact
6)	Manage Sector Resources

The monitor alert parameter (MAP) is used to inform traffic flow managers of potential capacity imbalances (FAA (2006c), Table 10). The MAP relates the average sector flight time to the peak number of aircraft occupying a sector analyzed at 15 minute intervals. If traffic is projected to exceed the MAP value then the sector is classified into a yellow or red alert and a notice is sent to the facility traffic manager. Red alerts occur when airborne flights are sufficient for a sector to violate capacity. Yellow alerts are used when additional flights that are currently on the ground are required for a sector to exceed capacity. The flow manager may respond to the alert by re-routing traffic, imposing a flow rate restriction by slowing aircraft down, or issuing a ground holding instruction (Leiden and Green, 2000).

The MAP value for each sector is adjusted dynamically based on current and projected conditions. Downward adjustment of the MAP value usually occurs in the case of a navigational aid (NAVAID) failure, severe weather, and/or communication interruptions (FAA,

Table 9: Sector Tactical (R-side) and Planning (D-side) Controller Responsibilities.

R-side	D-side
Ensure separation[a]	Ensure separation[b]
Initiate control instructions[c]	Initiate control instructions[d]
Operate and monitor radios	Operate interphones[e]
Accept and initiate automated handoffs	Accept and initiate non-automated handoffs[f]
Assist non-automated handoffs when required	Assist automated handoffs when required
Assist D-side coordination	Coordinate (including pointouts)
Scan radar and correlate with flight progress strip	Scan flight progress strip and correlate with radar
Computer entries	Manage flight progress strips

[a] Primarily using radar information

[b] Primarily using flight strips for flights entering sector. Flight strips are used as the method to separate aircraft when the radar display fails.

[c] To separate aircraft, implement dynamic traffic management initiatives, and restrict flow

[d] To upstream aircraft (outside sector) or to provide assistance to R-side

[e] System for communicating with controllers in adjacent sectors

[f] D-side must also make R-side aware of handoff

Table 10: Monitor Alert Parameter (MAP).

Average Sector Flight Time (min)	MAP Value
3	5
4	7
5	8
6	10
7	12
8	13
9	15
10	17
11 or greater	18

2006c). Sector controllers may also request reduced traffic flow since the MAP value does not consider complexity of the situation and its corresponding impact on workload. A scenario where a lower flow rate may be requested is the case of a planning controller spending a disproportional amount of time managing the paper flight strips and does not have time for conflict detection (Leiden and Green, 2000).

Weber et al. (2005) presents one of the few papers that considers changes to sector capacity during severe weather. Many existing models assume that sector capacity is a known deterministic quantity but do not specify how to estimate dynamic sector capacities during severe weather. Weber et al. develops a method that uses convective weather forecasts to estimate the percentage of jet routes or airways that are blocked. This percentage can then be used to estimate sector capacity.

2.11 Center TRACON Automation System (CTAS)

CTAS is a decision support system developed jointly by NASA Ames Research Center and FAA (Erzberger, 1995). The suite of tools assists controllers in planning and controlling air traffic within Air Route Traffic Control Centers (ARTCCs) and the Terminal Radar Approach Control (TRACON) airspace. The goals of the CTAS project are to increase system efficiency and reduce delays while maintaining or exceeding the current level of safety. Controller workload is also a concern for any decision support tool. CTAS is segregated into eight distinct software modules, or tools, that assist ATC personnel by increasing situational awareness using graphical displays. Two of these tools, Traffic Management Advisor (TMA) and User Request Evaluation Tool (URET), have been deployed as part of Free Flight Phase 1 in an effort to modernize the NAS. The remaining tools are subject to ongoing evaluation and may be deployed in Free Flight Phase 2.

2.11.1 Traffic Management Advisor (TMA)

The TMA tool advises controllers on the sequencing and scheduling of flights that are approximately 35 to 200 nautical miles from their destination airport (Wong, 2000). TMA is currently in operation at the Los Angeles, Oakland, Denver, Minneapolis, Fort Worth, Atlanta, and Miami ARTCCs. Traffic Management Coordinators (TMCs) define the oper-

ational constraints for aircraft arriving from an adjacent sector and departing aircraft from airports within the ARTCC. TMA implements the TMC constraints to optimally assign delay to flights. Corresponding schedule revisions, estimated times of arrival (ETA), and scheduled times of arrival (STA) are provided to controllers through a graphical display on a real-time basis at regular update intervals comparable to radar scanning rates. TMA does not consider congestion or delay effects for the computation of ETA. STA is computed considering the TMC scheduling and sequencing constraints, which additionally limit STA to be no earlier than ETA. ETA and STA are calculated at the following reference points: outer meter arc, meter fix, final approach fix, and runway threshold. The difference between ETA and STA at each of these reference points is the cumulative delay.

Aircraft are grouped by engine type, destination airport, and meter fix into "super stream classes" to assist the sequencing process. Within each of these classes aircraft are sequenced on a first-come first-served (FCFS) basis with consideration for meter fix ETA. However, the FCFS order can be overridden and additional constraints can be imposed using the TMA software. The algorithm is a standard sequencing procedure described in Wong (2000). Similarly, the scheduling algorithm is based on common operations research procedures. The emphasis here is placed on data organization and the problem approach philosophy since many solution procedures to the sequencing and scheduling problems are computationally efficient.

Aircraft scheduling, which follows the sequencing procedure described above, is used to calculate the STA for each flight and assign delay. It is notable that Collaborative Decision Making principles are not fully applied here and a philosophy of a centralized decision making authority is still used for both sequencing and scheduling decisions. Arrival slots at congested airports operate under CDM rules and, as such, TMA operates within airport CDM constraints.

TMA allocates aircraft to airport runways based on the current acceptance rate of the runway and whether the acceptance rate is changing or expected to change. The runway allocation algorithm attempts to reassign aircraft from runways with decreasing capacity rates to runways with a projected increasing capacity rate. The goal is to produce a smooth arrival rate to each runway by resolving capacity imbalances between runways such that

airport capacity is not underutilized.

Lee et al. (2000) performs a human factors study for the effects of TMA on controller workload. The study is based on anecdotal evidence and survey questionnaires of TMA users. TMA was perceived by controllers to reduce their workload in most instances. Sector controllers noted an overall minimal decrease in workload with no feature that specifically increased workload. Lee et al. detail how controllers interacted differently with CTAS, most notable was how specialized features were utilized. Also, the impact of the CTAS graphical user interface (GUI) on controller workload was detailed while noting positive usability results from managers and controllers.

2.11.2 Multi-Center Traffic Management Advisor (McTMA)

The Multi-Center Traffic Management Advisor currently being developed is expected to address the need for coordinated metering at further distances (300+ nautical miles) than single-center TMA around major airports (Landry et al., 2003). McTMA can be considered as the collaboration of single-center TMA systems. FAA is currently conducting an evaluation of McTMA as part of the Free Flight Phase 2 program with a prototype deployment for Philadelphia International Airport (PHL) expected in 2005. McTMA introduces a new level of complexity into the sequencing and scheduling of aircraft since communication between several ARTCCs must be coordinated. As such the McTMA developers found that calculating an exact or "optimal" solution requires too much intercommunication between ARTCCs. The ETA prediction accuracy at meter fixes is also a problem for aircraft far from final approach. To overcome these problems the McTMA architecture never fully solves the sequencing and scheduling problem. Instead a robust schedule is provided to controllers at each meter fix. Generally, loose constraints are generated at upstream meter fixes and passed to, or must be satisfied at, downstream meter fixes. In this way poor long-term information does not negatively impact the coordination of smooth arrival rates to runways.

2.11.3 Final Approach Spacing Tool (FAST)

The Final Approach Spacing Tool (FAST) assists controllers in sequencing landings, assigning aircraft to runways, and scheduling flights in TRACON airspace on final approach.

However, human factors studies have shown that controllers have difficulty recognizing FAST GUI graphics with current display technologies so a passive version of FAST with an advisory only status was implemented initially (Isaacson and Robinson, 2001). A promising review of passive FAST showed airport flows increased by 9-13% (Davis et al., 1997).

The FAST sequencer decomposes approaches into segments such that aircraft are ordered by segment. Starting with the outermost segment the FAST sequencer merges segments using an algorithm that preserves the relative order of aircraft. Conflict delay is added to trailing aircraft when a violation of minimum separation standards is projected. This conflict delay is approximated as the difference between the lead aircraft STA and the trailing aircraft ETA.

In addition to spatial conflicts, or violations of minimum separation requirements, FAST is able to de-conflict flights arriving out of sequence to reference points such as a meter fixes (Isaacson and Robinson, 2001). The conflicts are classified into cases based on relative aircraft heading at the potential conflict such as: in-trail, merging, diverging, or crossing paths. For each of these cases FAST uses a knowledge-based system that accounts for controller preference for de-conflicting flights.

A heuristic approach is used for runway allocation in FAST that initiates by the selection of a preferred runway for all aircraft. An ETA time window is defined to define aircraft eligible for reassignment to an alternate runway. Aircraft having an ETA within that time window are considered for switching to an alternate runway. After all eligible aircraft are considered the FAST runway allocation module selects the aircraft reassignment that results in the largest reduction in system delay. This process repeats until no further significant benefit is observed. Aircraft may only be reassigned once due to the nature of operations within the TRACON airspace.

2.11.4 En Route Descent Advisor (EDA)

The En Route Descent Advisor (EDA) assists en route sector controllers in transitioning aircraft from center to TRACON airspace. EDA automates the positioning of aircraft to the arrival-metering fix at TRACON boundaries with additional features that help reduce and resolve potential conflicts. The conflict resolution scheme, however, differs from that

of typical ATC controllers. Generally, controllers first ensure conflict-free trajectories then impose arrival rate constraints. EDA modifies this approach by instead metering flights upstream of the congested airspace and ignoring most conflicts until they may be more accurately modeled at a closer time horizon of approximately 10 to 20 minutes (Green and Vivona, 2001). This is in keeping with the philosophy of Erzberger et al. (1997) who stated that only highly probable, near horizon, predictable problems should be solved upstream of the potential conflict. Green and Vivona (2001) observed that the metering first approach tended to reduce the number of conflicts that needed to be resolved. Also, the redistribution of conflict detection and resolution from downstream sectors, where conflicts and metering violations occur, to upstream sectors was observed to improve system efficiency.

The common situational awareness of controller and flight plan intent is a critical part of EDA. Active aircraft plans, or the plan currently used by ATC personnel to control a specific flight, are available to all sectors through EDA with the restriction that only the current sector is able to modify this plan. Alternative, or provisional, aircraft plans can be suggested by any sector controller and evaluated by EDA towards the goals of metering conformance and prevention of conflicts in downstream sectors. The EDA conflict analysis algorithm analyzes each conflicting arrival pair. The algorithm iterates through a range of simultaneous changes to cruise speed and altitude attempting to minimize downstream conflicts while maintaining descent speed and TMA calculated scheduled time of arrival.

Green and Vivona (2001) noted the key benefits of EDA are increased efficiencies through cruise and descent trajectories as well as the flexibility in relaxing restrictions at meter fixes and arrival merge points. These restrictions were originally conceived to increase predictability and reduce arrival peaks, tasks that can be delegated to EDA.

2.11.5 User Preferred Routing (UPR) Conflict Probe

UPR is a CTAS tool used by en route (ARTCC) air traffic controllers to provide aircraft with a more direct or wind optimal flight path through a sector (Erzberger et al., 1997). Using UPR the jet airway network is no longer considered fixed and controllers are allowed more flexibility in routing decisions. UPR includes a conflict detection algorithm similar to the one previously described in Sherali et al. (2003). However, the algorithm is simplified

by only considering one deterministic flight trajectory during a stochastic conflict analysis. Recall that Sherali et al. generated several potential flight trajectories each with an associated probability during conflict detection in APCDM. Additionally, the trajectory error in UPR is represented by a 3-dimensional ellipsoid instead of the rectilinear shell of Sherali et al.. Another important distinction between the two approaches is that UPR predicts conflicts on a 20 minute time horizon as compared to the indefinite time horizon of APCDM. Both approaches include a threshold probability below which conflicts are not considered in subsequent analysis.

A conflict resolution algorithm is also implemented in UPR that proceeds as follows. First a conflict detection iteration estimates the pre-resolution probability for violation of minimum separation standards. Then the ATC controller specifies the maximum post resolution conflict probability for each aircraft pair. Several iterations are performed on trajectory parameters to achieve a set of candidate solutions. A final candidate is selected that minimizes the maneuver time to achieve the specified post-resolution conflict probability. Erzberger et al. notes that near optimal solutions are achievable by this heuristic and more thorough procedures, such as those based on control theory, are generally not necessary.

2.11.6 Direct-To

Direct-To is an advisory system for air traffic controllers to identify aircraft that can reduce flight times by at least one minute by flying directly to a downstream meter fix (Erzberger et al., 1999). This CTAS tool utilizes the National Weather Service atmospheric model to aid in the prediction of wind optimal routes for eligible flights. Direct-to examines each flight every 12 seconds and calculates, based on a database of acceptable direct-to fixes, if a more direct path can save flight time. Controllers are then able to issue a direct-to clearance to pilots. Currently direct-to is not used around major hub airports due to complexities caused by metering restrictions.

3 EXISTING APPROACH VALIDATION

During airspace restriction scenarios FAA currently uses the National Severe Weather Playbook for strategic aircraft routing. The Playbook is generally straightforward for traffic managers and controllers to implement but may not always result in the best, or close to the best, strategy for handling en route air traffic during congestion events. One proposed alternative approach of Sherali et al. (2003) is an Airspace Planning and Collaborative Decision-Making Model (APCDM) using integer programming techniques for selecting aircraft routes. APCDM has the potential advantage of accounting for a wider range of scenarios and real time conditions but is generally considered too computationally expensive for practical deployment. A more detailed description of these two methods is described in the literature review of Section 2.

Another weakness of APCDM is that the model has not been validated against the Playbook. The RAMS airspace simulation software (ISA Sofware Ltd., 2004) is used to compare the Playbook and APCDM approaches under a range of en route conditions. The airport to airport simulation scenarios focus on en route conditions. More specifically, each airport is represented as a point without regard for airport arrival and departure capacities or taxi times to and from the gate. The end goal of this process is an unbiased comparison of the Playbook and APCDM management strategies during severe weather and other en route airspace restrictions.

3.1 Scenario Selection

Simulations are computationally time consuming and consequently relevant date, times, and locations are required to specify appropriate scenario conditions. The FAA has developed 7 days of typical weather experienced in the National Airspace System (Table 11). Times and locations of excessive delays are identified using FAA Operations and Performance Data (FAA, 2005). These times and locations are confirmed visually using time stamped NEXRAD reflectivity images (Figure 9). However, the selection process is somewhat qualitative in that airport delay effects and en route delay effects are difficult to separate. Table 12 details the scenarios under consideration.

Table 11: FAA Weather Days.

1) February 19, 2004
2) May 10, 2004
3) June 10, 2004
4) June 27, 2004
5) July 27, 2004
6) September 26, 2004
7) October 22, 2004

Table 12: En Route Simulation Analysis Scenarios.

	Scenario 1	Scenario 2	Scenario 3	Scenario 4
Date	Feb. 20, 2004	May 11, 2004	May 11, 2004	June 11, 2004
Time (UTC)	14:00-15:00	13:00-14:00	18:30-20:30	12:15-14:00
Weather	Snowstorm	Thunderstorms	Thunderstorms	Thunderstorms
Affected Center	Chicago	Kansas City	Indianapolis	Chicago
Other Centers In Analysis		Chicago Minneapolis Denver Albuquerque Fort Worth Memphis	Cleveland Chicago	Indianapolis Cleveland
Number of Flights	568	1553	878	641
Number of Surrogates	6	6	10	10

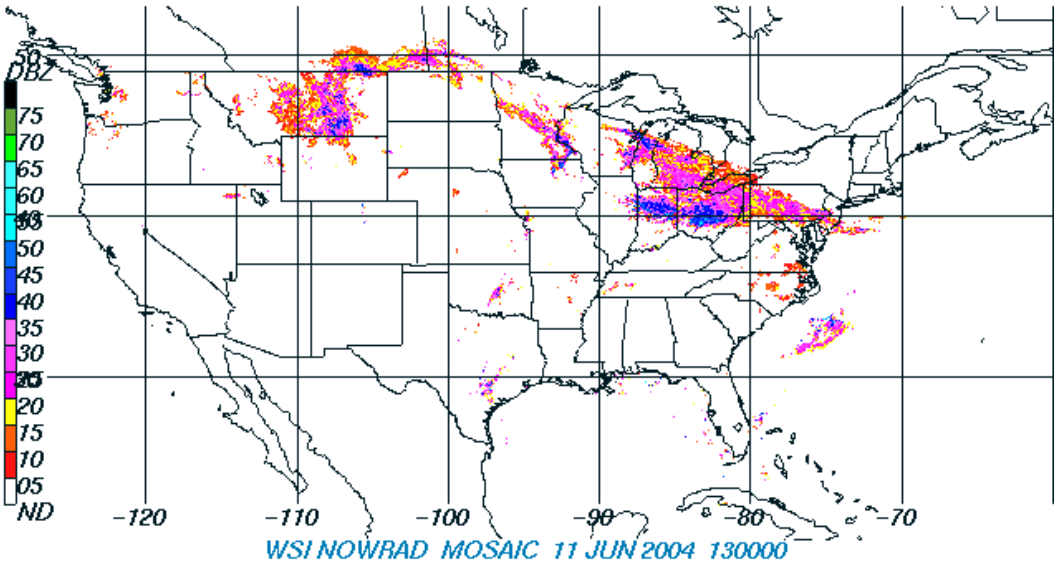

Figure 9: NEXRAD Reflectivity June 11, 2006 13:00 UTC (NOAA Research, 2006).

3.2 Data

Historical flight information archived from the Enhanced Traffic Management System (ETMS) is used as a basis for this analysis (Volpe National Transportation Systems Center, 2002). ETMS has been developed to facilitate the transfer of data between FAA facilities and airlines using a range of messages (Table 13). Figure 10 illustrates the process of combining ETMS data with aircraft, passenger, airspace, and weather data to produce input files for RAMS. A detailed description of the ETMS data files is included in Appendix A.

Table 13: ETMS Messages used for Simulation Analysis.

Message Type (Prefix)	Description
Flight Amendment (AF)	Revisions to flight plan and departure time
Flight Plan (FZ)	Planned route of flight through NAS
Flight Departure (DZ)	Wheels off time for flight
Center Crossing (UZ)	Time (actual) of flight crossing en route center
Flight Cancellation (RZ)	Cancellation of planned flight
Flight Track (TZ)	Aircraft position and speed (radar)
Flight Arrival (AZ)	Time of flight arrival to destination airport

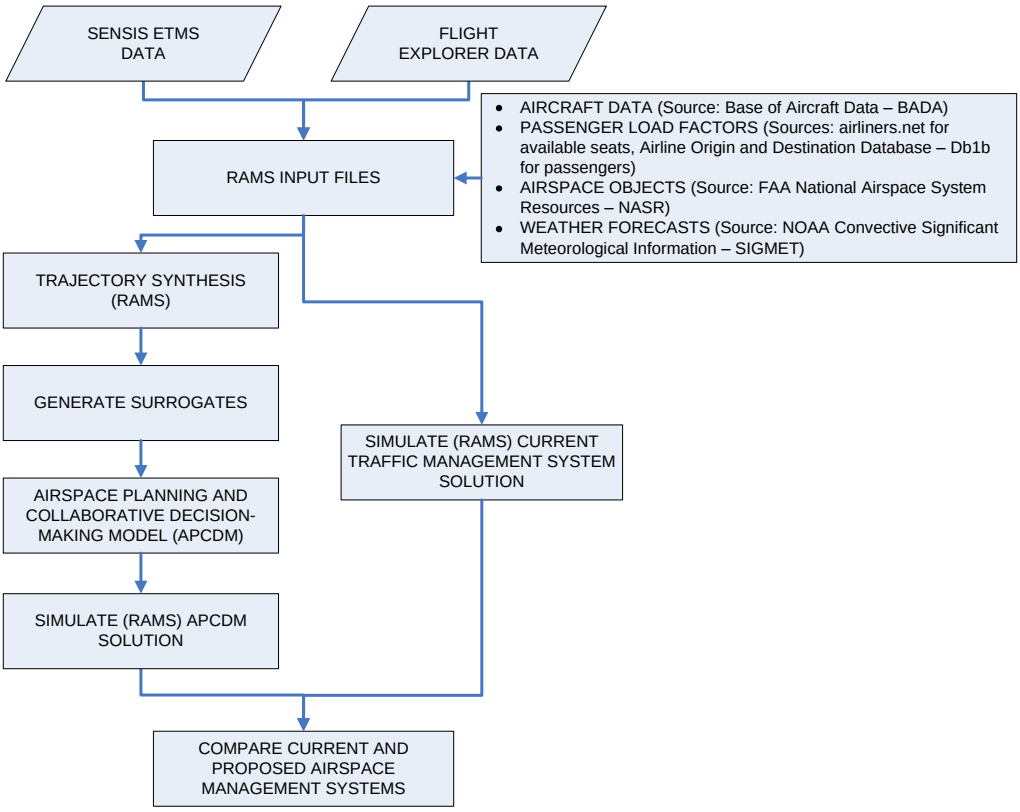

Figure 10: Overview of RAMS Simulation Data Preparation and Comparison Process.

Eurocontrol Base of Aircraft Data (BADA) is the source for aircraft operational characteristics (Eurocontrol, 2003). BADA performance tables detail airspeeds and fuel flows for the climb, cruise, and descent portions of flight at various flight levels. Information specifying thrust, drag, and fuel consumption are also included but not considered here. Eighty-seven (87) aircraft are modeled in BADA, a number that can be expanded to include 267 by considering 180 substantially similar aircraft. These aircraft represent over 95% of the aircraft types included in the ETMS files. Substitution aircraft for unknown types is not expected to significantly affect the analysis since both the Playbook and APCDM flights use the same substituted aircraft types.

APCDM does not explicitly consider weather conditions, however it is expected that airlines provide surrogates capable of avoiding severe weather. The Collaborative Convective Forecast Product (CCFP) 0 to 2 hour forecast (Section 2.4) is used in the flight planning stage to select candidate flight trajectories. The actual, or historically observed, weather conditions

may be different from the CCFP forecast and are obtained from NEXRAD reflectivity and echo tops observations (National Climatic Data Center, 2005). Echo tops refer to the ceiling of a weather system. ETMS flight track (Table 13) information is used to estimate areas that pilots avoid for each weather system. During simulation of actual weather conditions a restriction zone is developed that prevents flights from entering severe weather conditions (Figure 11).

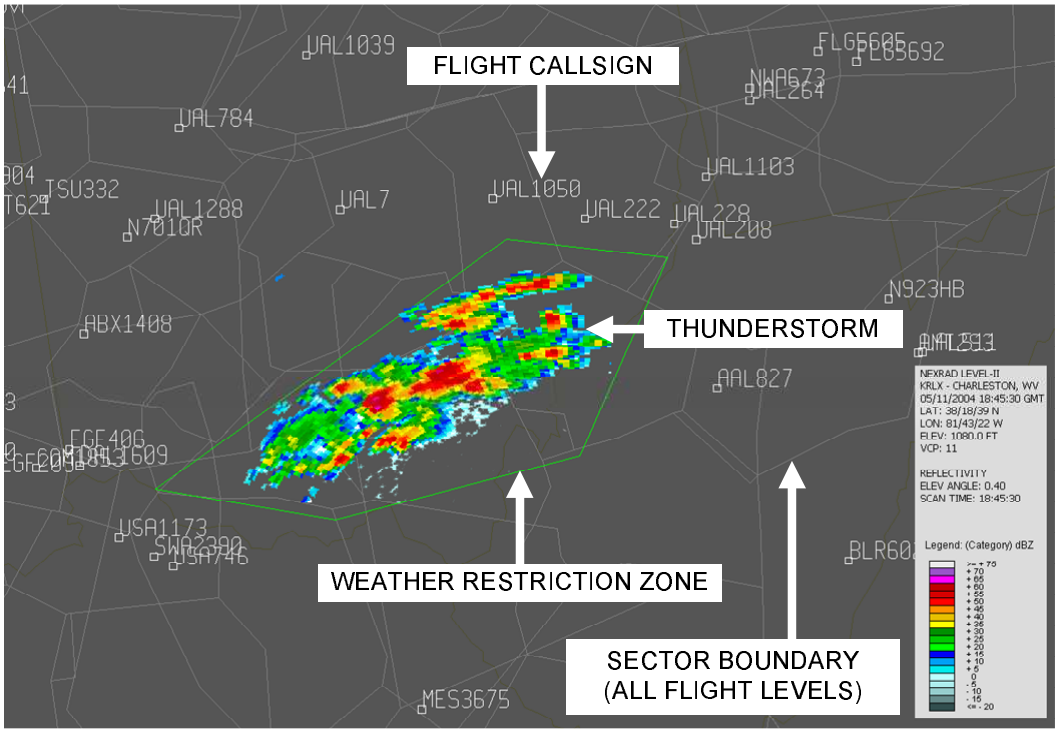

Figure 11: Restriction Zone Approximation of Pilot Avoidance of Severe Weather.

3.3 Trajectory Synthesis

Airlines and other NAS users provide flight trajectories to the FAA via ETMS flight plan (FZ) and flight amendment (AF) messages. Each message references, by an identifier, airspace objects such as airports, navigational aids, fixes, air route traffic control centers (ARTCCs), and others as specified in the National Airspace System Resource (NASR) database (FAA, 2004b). The ETMS messages include the expected departure time, arrival time, and cruising flight level (altitude). However APCDM requires the time and altitude at each trajectory

waypoint which is not provided.

The process of using aircraft performance characteristics to estimate the altitude and time for a flight plan is generally referred to as trajectory synthesis. CTAS has a trajectory synthesis mechanism to estimate sector loads and probe for conflicts (Section 2.11). RAMS is used for trajectory synthesis in this analysis based on aircraft characteristics and waypoint information.

3.4 Flight Surrogate Generation

The performance of APCDM in terms of solution quality is dependent on the alternative trajectories provided. Six surrogates were considered initially (Table 14) with the addition of four surrogates having no noticeable difference on solution quality. The restriction surrogate (Figure 12) provides an alternative trajectory, calculated using the RAMS avoidance rule base, around congested airspace.

Table 14: Candidate Alternative Trajectories (Surrogates) for APCDM.

Surrogate	Description	Scenarios
Nominal	Unmodified ETMS flight plan message	1-4
+15 min	Flight departs 15 minutes after scheduled departure	1-4
+30 min	Flight departs 30 minutes after scheduled departure	1-4
-20 FL	Cruising flight level is reduced by 2000 ft (20 FL)	1-4
Restriction	Alternative trajectory around congested areas	1-4
Cancellation	Flight is cancelled	1-4
+3 min	Flight departs 3 minutes after scheduled departure	3,4
+6 min	Flight departs 6 minutes after scheduled departure	3,4
+9 min	Flight departs 9 minutes after scheduled departure	3,4
+12 min	Flight departs 12 minutes after scheduled departure	3,4

3.5 Results and Discussion

APCDM found a solution for Scenarios 1 to 3 (Table 12) but was unable to resolve all conflicts on the arrival path to Chicago O'Hare during Scenario 4. Default parameters (Table 15) are

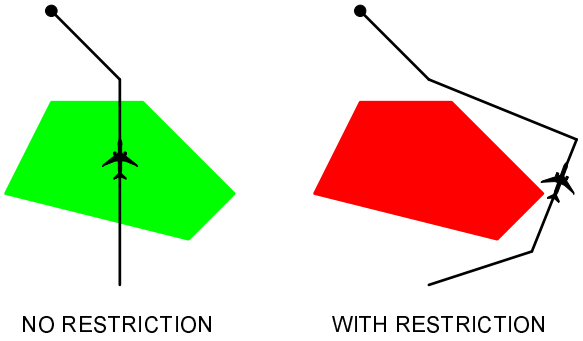

Figure 12: Alternative Trajectory (Surrogate) based on Restriction.

used for the first three scenarios and modified in an attempt to gain feasibility for the fourth scenario.

Table 15: Default Parameters used for APCDM Calculations.

Parameter	Value
Value of Time	$26.70/hour (DOT, 1997)
Cancellation Cost	1440 min. (1 day)
Seats Offered per Flight	Official Airline Guide OAG Worldwide Limited (2004)
Passenger Load Factor	0.8
Encounter Probabilities (PAEM)	Defaults provided
Prep Buffer	10 min.
Maximum Simultaneous Conflicts	5
Maximum Sector Occupancy	15 or 17
Equity Parameters	Defaults

Average flight delay is a key indicator of performance from the FAA perspective. The focus here is en route delay, that delay component experienced by flights in the en route airspace (altitudes exceeding 18,000 feet) due to severe weather avoidance and air traffic control actions required to maintain minimum separation of aircraft. Figure 13 reveals that APCDM does not produce noticeably smaller delays per flight. A similar trend for fuel use (kg) and delay cost ($) is observed.

Conflicts (Figure 14) and flights monitored by a sector controller (Figures 15 and 16)

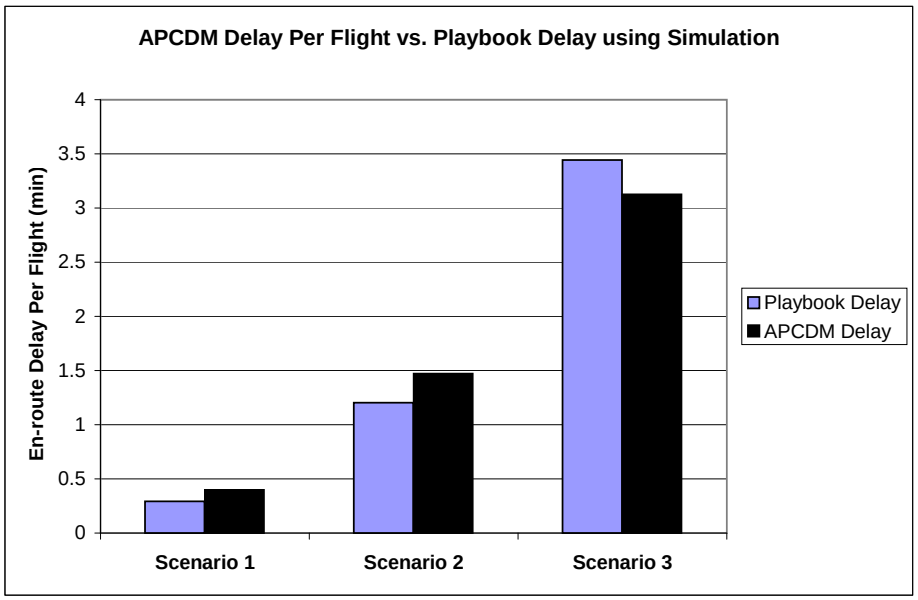

Figure 13: APCDM vs. Playbook (Historical) Delay.

are the primary air traffic controller workload metrics. Conflicts refer to the number of conflict resolution instructions to pilots by air traffic controllers to prevent loss of minimum separation. APCDM does not decrease the aggregate number of conflict resolution actions required for the three scenarios analyzed. The peak monitoring workload is reduced for the less congested Scenarios 1 and 2 but is unchanged for the congested Scenario 3.

APCDM seeks to provide an equitable solution to all NAS users during airspace restriction conditions. This goal is achieved though the results are difficult to quantify and may be a result of lack of system wide improvement. APCDM does not seem to significantly improve congestion or controller workload. The poor performance of APCDM reduces the motivation for further analysis since APCDM should improve operations during every potential situation. Reasons for the poor results are problematic to isolate but those criticisms described in Section 2.9 are candidates. A potential drawback for this analysis is the lack of suitable alternative trajectories which APCDM relies on for improvement. Generally, if APCDM is provided with more informed trajectories that avoid congested airspace then superior results can be expected.

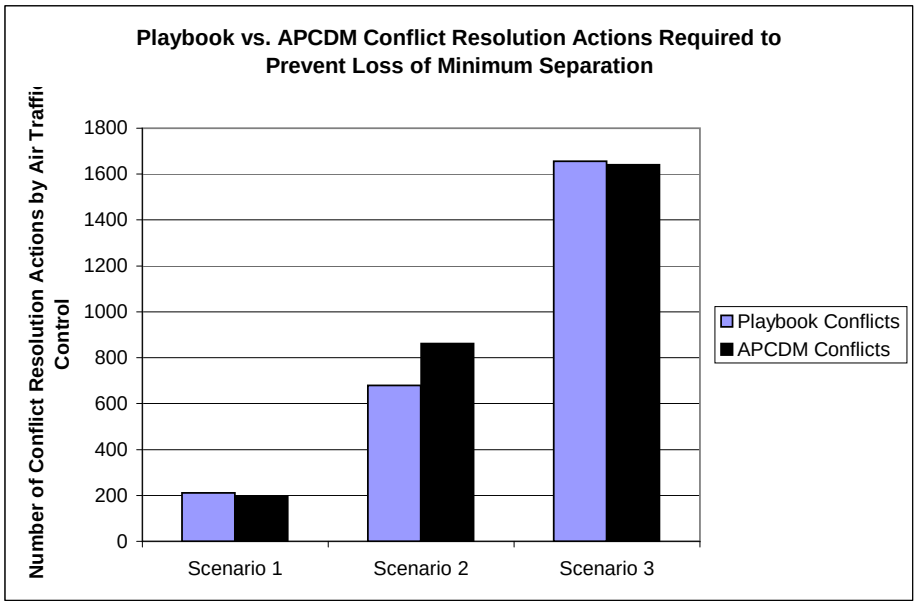

Figure 14: APCDM vs. Playbook (Historical) Conflict Resolution Actions.

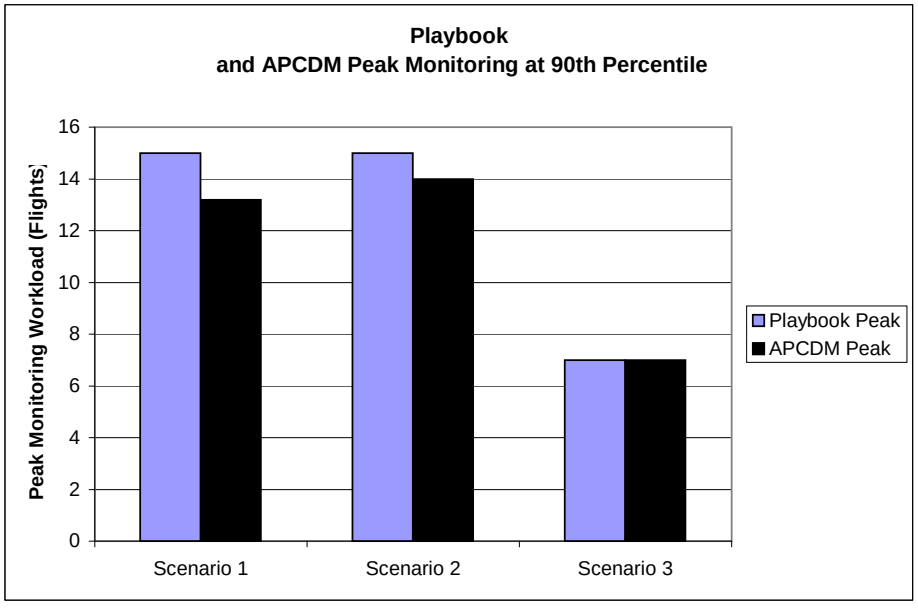

Figure 15: APCDM vs. Playbook (Historical) 90^{th} Percentile Peak Flights Monitored per Sector.

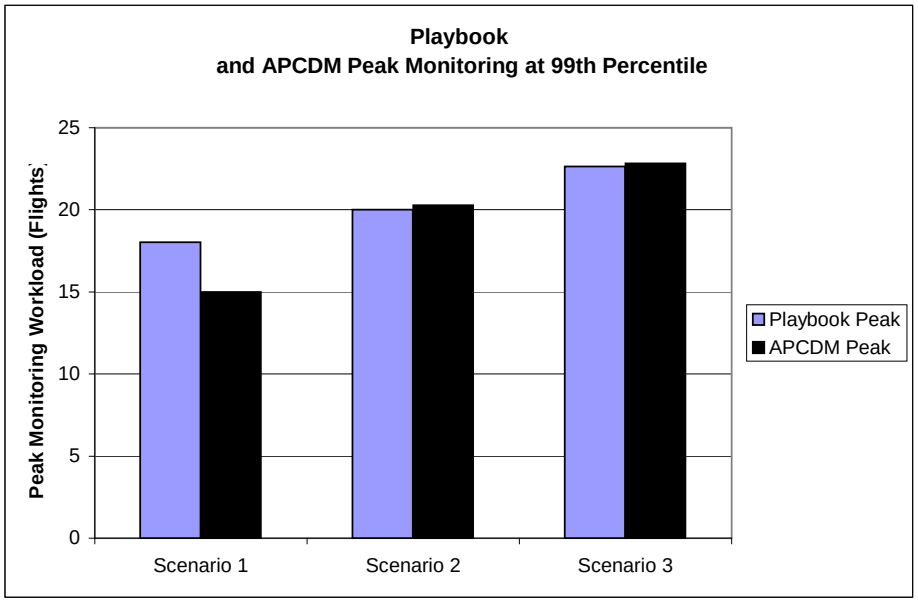

Figure 16: APCDM vs. Playbook (Historical) 99^{th} Percentile Peak Flights Monitored per Sector.

4 MODEL FRAMEWORK

A better airspace planning model should explicitly consider stochastic airspace elements. A model that adapts to poorly forecasted information such as departure time, en route travel time, and en route aircraft position is highly desirable. The model presented here is dynamic in its ability to plan for changing demand and capacity conditions.

The proposed approach to airspace planning (Figure 17) includes a system-wide contingency planning feature. Contingency planning is aimed to address the problems of a single solution based on expected conditions since if the weather conditions change, as often happens, there can be chaos for air traffic controllers who now lack a suitable plan. Alternatively, if the forecasted weather system does not develop as expected then aircraft may be unnecessarily rerouted creating wasted airspace capacity resulting in unnecessary user costs.

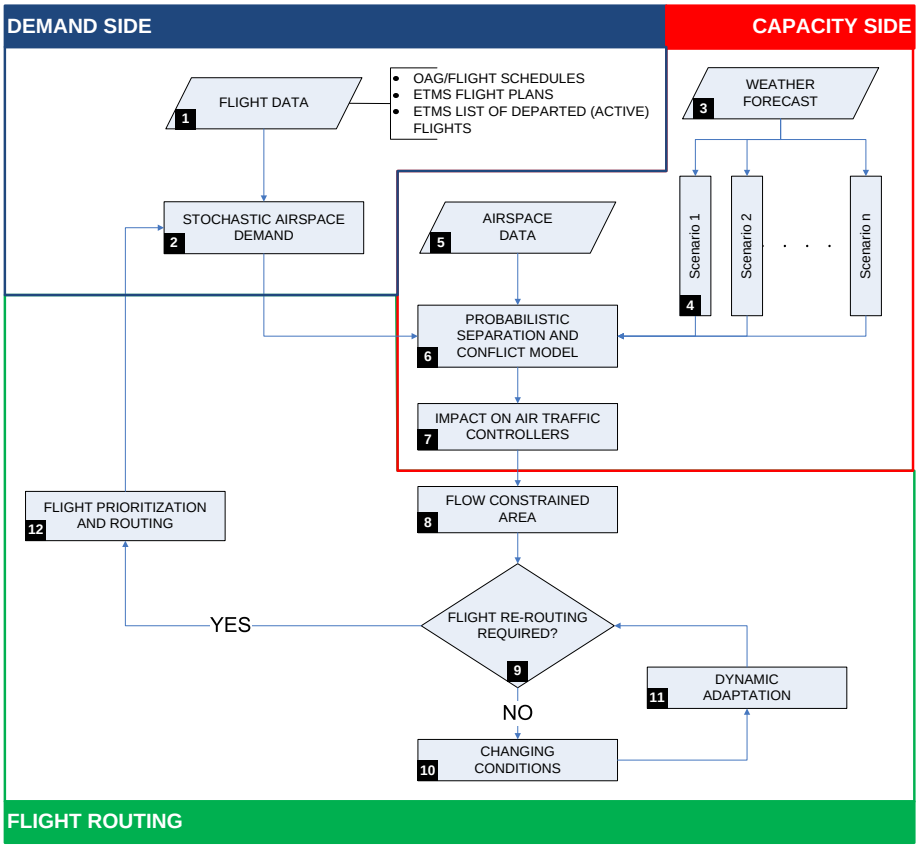

Figure 17: Proposed Capacity Restriction Response Framework.

The relationship between the model components shown in Figure 17 and referenced by numbered boxes is as follows. The demand for en route airspace is calculated using the flight schedules contained in the Official Airline Guide (OAG) until a flight plan is received into the Enhanced Traffic Management System (EMTS) (box 1). Demand information is updated by ETMS when a flight departs. This information is used to develop demand for en route airspace considering random departure and en route conditions (box 2).

Capacity analysis of the en route airspace proceeds by examining the convective weather forecast 1-2 hours in advance of the capacity constrained event (box 3). The current state of convective weather forecasting is such that these forecasts have a high degree of uncertainty. To account for this uncertainty a range of weather scenarios are generated so that alternative conditions can be considered (box 4). The reduction in available airspace is extracted from each of the weather scenarios and combined with airspace demand information (box 2) and airspace sector geometry and restrictions (box 5) to produce estimates of separation and conflict rate within the airspace (box 6). In this dissertation stochastic geometry and spatial point processes are used to develop estimates of separation and conflicts. The workload imposed on air traffic controllers is then estimated using predicted separation and conflict rates (box 7).

Flights are selected for re-routing based on a flow constrained area (box 8). This area is calculated using the probability that a region of airspace will exceed a probability threshold for congestion during a period of time. Flights that pass through the flow constrained area are candidates for re-routing (box 9) based on prioritization of flights (box 12). As weather and demand conditions change (box 10) the traffic flow managers must be able to adapt to prevent demand-capacity imbalances (box 11).

5 STOCHASTIC AIRSPACE DEMAND

This section describes extensions to current stochastic airspace demand models to include pre-departure uncertainty, en route traversal uncertainty, and route uncertainty. A method to combine departure time uncertainty and en route traversal time uncertainty is presented and applied to one day of historical airspace conditions to quantify the benefit of a probabilistic approach.

5.1 Departure Uncertainty

Previous work in the area of sector demand estimation has noted several factors that lead to errors in sector demand (Wanke et al., 2004; Lindsay et al., 2005; Krozel et al., 2002). The work presented here will focus on pre-departure, sector traversal, and route uncertainty. Pre-departure uncertainty is the difference between the proposed wheels-off time at the origin airport and the measured departure time as recorded in ETMS. The ETMS system does not provide the most accurate historical prediction of wheels-off time, however errors of a few minutes are considered negligible in the context of this analysis.

The quantity of interest is the departure prediction error and not deviation from the schedule so the lateness of a flight is not what is being measured. The following is a partial list of factors that can result in poor estimations of departure time: aircraft arriving late from a previous leg, unavailable gates from the previous leg, crew arriving late, aircraft servicing, de-icing operations, runway direction reversals, taxiway availability, etc.

The procedure to calculate departure uncertainty begins by collecting all relevant messages from the ETMS historical data including the flight schedule (FS), flight plan (FZ), flight amendment (AF), control departure time (CTRL), and flight cancellation (RZ) messages. The Python scripts for these calculations are included in Appendix B starting on page 136.

The analysis days for this study are shown in Table 16 from which 1,238,730 departure observations are extracted. Information from the previous day is also used to obtain full flight plan and schedule information. Gate push-back times are obtained from the FAA Airline Service Quality Performance (ASQP) database (FAA, 2005). Definitions for departure time

are shown in Table 17.

Table 16: Study Analysis Days.

Day	Year(s)
February 19	2000-2005
May 10	2000-2005
June 11	2004-2005
June 27	2004
July 14	2005
July 27	2000-2005
August 29	2005
September 26	2000-2004
October 23	2004
December 1	2001-2002
December 3	2000
November 28	2004
November 30	2003

Table 17: Departure Time Definitions.

Notation	Definition
ETMS modeled departure time	Gate pushback time + ETMS modeled taxi time
ETMS modeled taxi time	Moving average of last five taxi times for that flight
Wheels-off time	Estimated runway-off time for flight from ETMS message
Departure time prediction error	Wheels-off time - ETMS modeled departure time
Look-ahead time	ETMS modeled departure time - current time

For example if a flight plan message is received at 0200Z with a modeled departure time of 0330Z, then a subsequent flight amendment is received at 0250Z then only an error observation corresponding to a 60 minute look-ahead time is recorded for the 0200Z message.

The analysis proceeds by attempting to find structural variance in the prediction error data. Exploratory analysis strongly suggests the existence of non-constant variance across

variables, otherwise known as heteroscedasticity. A modified least squares regression procedure is used since errors are non-normal and right-skewed even under a logarithmic transformation. Another method that accounts for non-constant variance is the class of generalized autoregressive conditional heteroscedasticity (GARCH) models most suitable to time series analysis but with limited applicability to this problem (Tsay, 2005).

The modified regression procedure is as follows. The regression form of Equation 24 shows a response variable matrix $\mathbf{Y}$ to be a function of two independent random variables $\mathbf{X}$ and $\mathbf{E}$ and a coefficient matrix $\mathbf{B}$. If the variance is constant then $E \sim N(0, \sigma^2)$ is independent of $\mathbf{X}$. This model is extended by allowing the variance to be a function of $\mathbf{X}$ as shown in Equations 25 and 26 (Greene, 2003). An exponential link function is used in this analysis but others may be substituted.

$$\mathbf{Y} = \mathbf{BX} + \mathbf{E} \tag{24}$$

$$\mathbf{E} \sim \mathbf{N}(\mathbf{0}, \sigma^{\mathbf{2}}\lambda(\mathbf{X})) \tag{25}$$

$$\lambda(\mathbf{X}) = exp(\mathbf{\Gamma X}) \tag{26}$$

This type of regression on the variance does not eliminate the non-normality problem but it does allow an investigation into the conditions that lead to larger variance. A total of 26 explanatory variables are considered representing flight, airline, airport, and weather conditions. A description of the data file used in this analysis is included in Appendix B starting on page 173.

The model is calibrated using SAS (SAS Institute Inc., 2003) with coefficients for the 14 selected variables presented in Table 18. The SAS script is included in Appendix B starting on page 176. The exponential of the coefficients is also shown since the coefficients must be transformed back and used as a multiplier effect. All variables are significant at the 5% level though the test for statistical significance is somewhat questionable in this case due to the non-normality of the response variable. A logarithmic transform of the error response at a 30 minute look-ahead time (E_{LAT30}) is used to better approximate normality as shown in Equation 27.

$$Y = log(E_{LAT30} + 60\,minutes) \tag{27}$$

Table 18: Regression Coefficients for Mean and Variance of Departure Time Prediction Error at a 30 Minute Look-Ahead Time.

	Coefficient for Mean Estimate			Coefficient for Variance Estimate		
	B	$exp(B)$	$S.E.^a$	Γ	$exp(\Gamma)$	$S.E.^a$
Intercept	4.175	65.040	0.0009	0.277	1.319	0.0005
Depart from a large hub	0.014	1.014	0.0009	-0.019	0.981	0.0051
Depart from a medium hub	-0.004	0.996	0.0011	0.038	1.039	0.0060
Depart from a small hub	-0.064	0.938	0.0013	-0.303	0.739	0.0059
Depart from a non-hub	-0.004	0.996	0.0011	-0.170	0.844	0.0066
Departing airport operating under instrument conditions (IMC)	0	1		0.284	1.328	0.0025
A large carrier (top 25 by operations) departing from a large hub airport	0	1		-0.367	0.693	0.0030
A large carrier (top 25 by operations) departing from a medium hub airport	0	1		-0.681	0.506	0.0048
A large carrier (top 25 by operations) departing from a non-hub airport	0	1		-0.048	0.953	0.0088
A large carrier (top 25 by operations) departing from a foreign airport	0	1		-0.313	0.731	0.0061
A small carrier (not top 25 by operations) departing from a small airport and arriving to a large hub	0	1		0.379	1.461	0.0048
If flight plan is amended	0	1		0.100	1.105	0.0018
If convection is forecasted to impact this flight (origin airport, en route, destination airport)	0	1		0.047	1.048	0.0022
If flight has been cancelled and reactivated	0	1		0.198	1.219	0.0039
If flight has been both amended and cancelled and reactivated	0	1		0.408	1.504	0.0052
Sample size	1,130,874					
Log Likelihood	-83,918.2					

[a] Standard Error

Since the error could not be completely transformed to normality a categorical analysis of the distribution of errors is considered based on the results of the regression analysis. The first of the groupings uses the departing airport type of the flight. A box-and-whisker diagram of the errors (Figure 18) shows the 25th percentile, median, and 75th percentile

of the errors as a box. The difference between the 75th percentile and the 25th percentile is known as the interquartile range (IQR) and is used to mark the largest and smallest observations that are "valid" as whiskers. A data point is considered valid if it is less than 1.5(IQR) from the box. Outliers are indicated by a "+". The notable characteristics of the diagram are that the median is relatively constant between airport types, all the distributions are right-skewed (positively skewed), variance increases as airport size decreases, and there are numerous outliers for each distribution, which is the reason for the solid red line. Large variance for smaller airports may seem counter-intuitive but the type of airline operating at these airports has an impact.

Regional air carriers and general aviation operators generate a significant portion of the traffic at smaller airports. It is not known why these operations exhibit more variance but the following potential explanations are suggested. Smaller airports are less congested and smaller airlines may feel less pressure to provide accurate information since demand to capacity imbalances are rare. Also, smaller airports act as "feeders" to large airports and do not perform hubbing type operations that require more strict adherence to a schedule. The schedule padding, which refers to the time in excess of the minimum gate-to-gate operation time, may also be larger to ensure connections to routes with higher passenger levels.

A categorical grouping that includes factors in addition to departing airport size is shown in Figure 19 with a corresponding box-and-whisker plot in Figure 20. Appendix B contains the SAS script for the tree diagram (page 179) and the MATLAB script for the box-and-whisker plot (page 184). The clustering procedure covers all cases and the order is generally as follows: amendment, cancelled, forecasted convection, airport type, if the carrier is one of the top 25 carriers by operations, and arriving airport size. Clusters are ranked by mean error then by variance so that cluster 1 has the lowest mean and variance while cluster 10 has the highest mean and variance. The highest variance is for flights that have amendments or that have been cancelled and reactivated. By separating smaller carriers from larger carriers this analysis shows that larger carriers have lower variance than smaller carriers and smaller airports have lower variance than larger airports when corrected for carrier type. However, since smaller carriers dominate smaller airports we get the results shown in Figure 18.

An attempt is made to generalize the errors to a probability distribution. However,

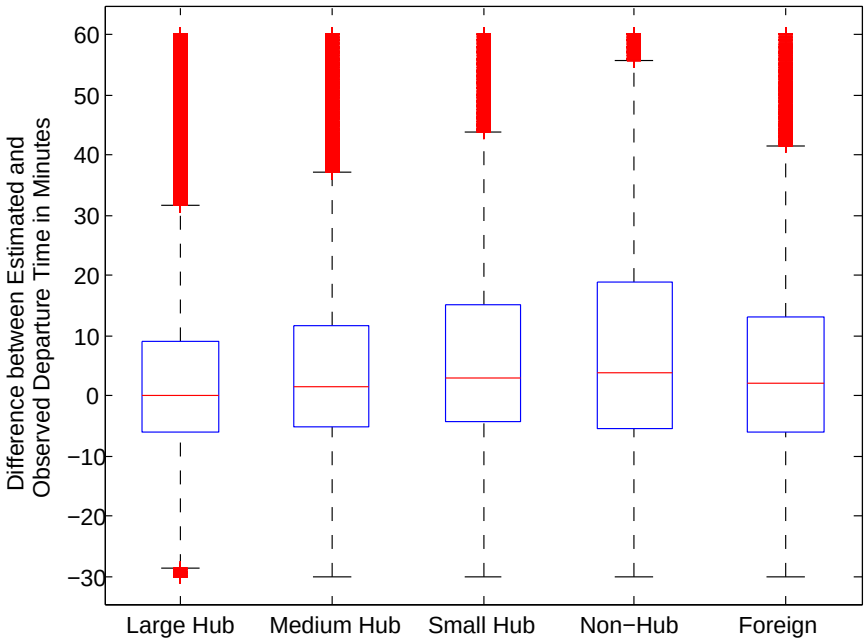

Figure 18: Box-and-Whisker Diagram for Departure Time Prediction Errors at 30 Minute Look-Ahead Times by Departing Airport Type.

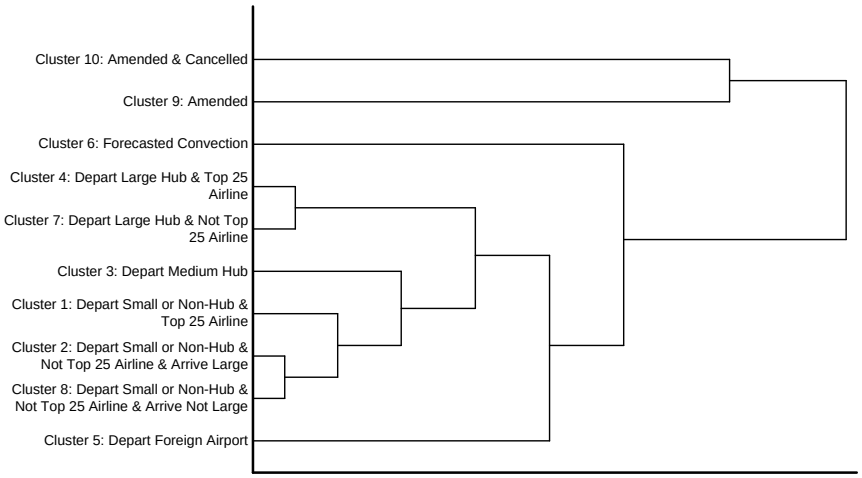

Figure 19: Tree Diagram for Clustering.

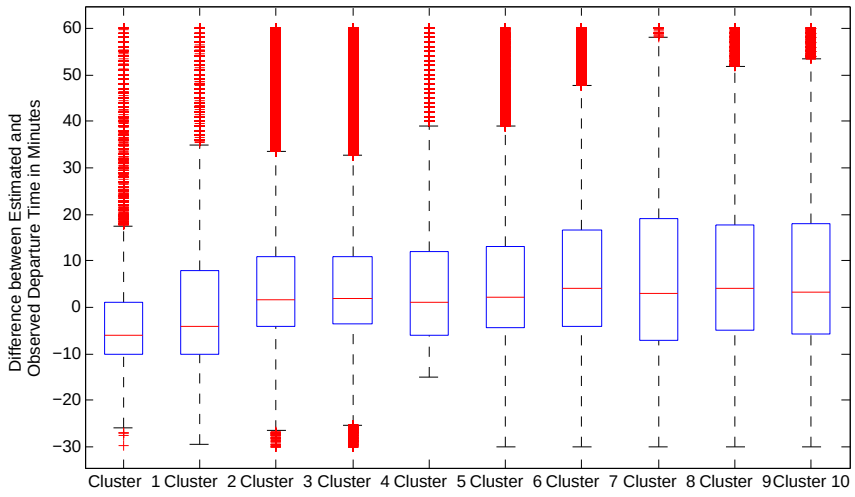

Figure 20: Box-and-Whisker Diagram for Departure Time Prediction Errors at 30 Minute Look-Ahead Times by Cluster.

since the error is right-skewed and peaked around 0 the standard distributions are poor approximations (Figure 21). Histograms are constructed and compared to the lognormal for each of the two groupings considered here: airport type and clustered data. The SAS script to create the histograms is included in Appendix B on page 186. For each of these histograms the lognormal approximations are significantly different from the observed empirical distribution.

In kernel smoothing a probability mass, such as a normal or other symmetric density function, is placed at each data point. The equations to place the probability mass are straightforward. Begin by specifying a kernel that satisfies Equation 28. In this case the standard normal distribution is chosen $K(x) \sim N(0,1)$. The density at each value is estimated by summing all kernels as detailed in Equation 29 where n is the number of samples and h is the bandwidth. The optimal bandwidth parameter is a function of sample size and variation of the distribution. Larger sample sizes permit a smaller bandwidth while larger variances require increased bandwidth. Table 19 provides results of the kernel smoothing for the two grouping methods in terms of the bandwidth smoothing parameter h.

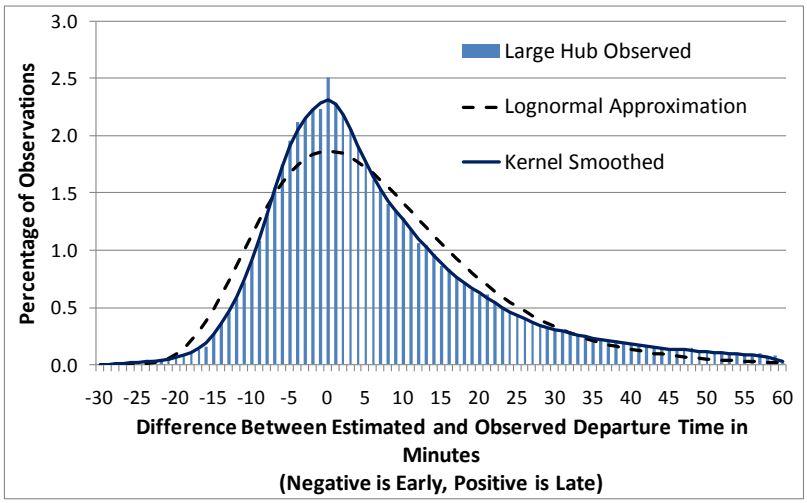

Figure 21: Histogram of Departure Time Prediction Errors at a 30 Minute Look-Ahead Time Frame for Large Hub Airports for 30 days in 2000-2005. A Lognormal Approximation and Kernel Smoothed Distribution are also Displayed Illustrating a Poor and Good Fit Respectively.

$$\int K(x)dx = 1 \tag{28}$$

$$\hat{f}(x;h) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right) \tag{29}$$

5.2 Sector Traversal Time Variation

Another source of randomness in the estimation of sector demand is the traversal time through the sector. Controller actions such as: speed changes, vectoring, issuing a holding pattern, or the clearance of a more direct route may cause the flight to spend more or less time in the sector than planned. The approach used compares the planned sector flight time obtained through simulation to the observed sector traversal time from the processed ETMS radar track data (TZ). The scope of the analysis includes the following east coast air route traffic control centers (ARTCCs): Chicago (ZAU), Indianapolis (ZID), Atlanta (ZTL),

Table 19: Bandwidth Estimation Results for Kernel Smoothing.

Data Set	Sample Size	Bandwidth h (minutes)
Large Hub	471,003	0.922
Medium Hub (Cluster 3)	171,702	1.067
Small Hub	121,814	1.080
Non-Hub	247,909	1.455
Foreign (Cluster 5)	122,147	1.358
Cluster 1	99,318	0.786
Cluster 2	48,049	1.637
Cluster 4	346,769	0.890
Cluster 6	169,169	1.255
Cluster 7	111,859	1.612
Cluster 8	48,049	1.637
Cluster 9	87,090	2.260
Cluster 10	30,423	3.786

Jacksonville (ZJX), Miami (ZMA), Washington (ZDC), Cleveland (ZOB), New York (ZNY), and Boston (ZBW).

To obtain planned sector traversal times the most recent flight plan or amendment before the actual departure is extracted from ETMS data. The flight plan data is converted into a format suitable for the RAMS Plus airspace simulation software (ISA Sofware Ltd., 2004). Other information including aircraft performance, airport locations, navigational aids (NAVAIDs), fixes, airways, standard terminal arrivals (STARs), and departure paths are also converted to the RAMS format. Aircraft performance uses EUROCONTROL's Base of Aircraft DAta (BADA) (Eurocontrol, 2003) which is different from the ETMS system aircraft performance models (Volpe National Transportation Systems Center, 2002). The largest source of uncertainty in aircraft performance modeling is the prediction of aircraft weight. In this analysis we assume a nominal, or average, weight for each flight based on the three aircraft mass categories contained in BADA: low, nominal, and high. Each of the flight plans are then simulated to obtain the time of sector entry, time of sector exit, and sector traversal

time. The air traffic controller functionality of RAMS is turned off so there is no conflict resolution for flights predicted to violate minimum separation standards.

The ratio of observed sector traversal time to planned sector traversal time, which is obtained from processing the RAMS output files, is examined for structure. A plot of the standard deviation of the ratio of observed to planned sector traversal times by planned sector traversal time and observed airspace density (Figure 22) shows that the planned traversal time through the sector has a larger effect than the observed airspace density. This does not mean to suggest that airspace density has no effect since sectors with shorter traversal times are typically more congested than those with longer traversal times. The assertion here is that flown traversal time is mostly impacted by planned time for a flight to cross a sector.

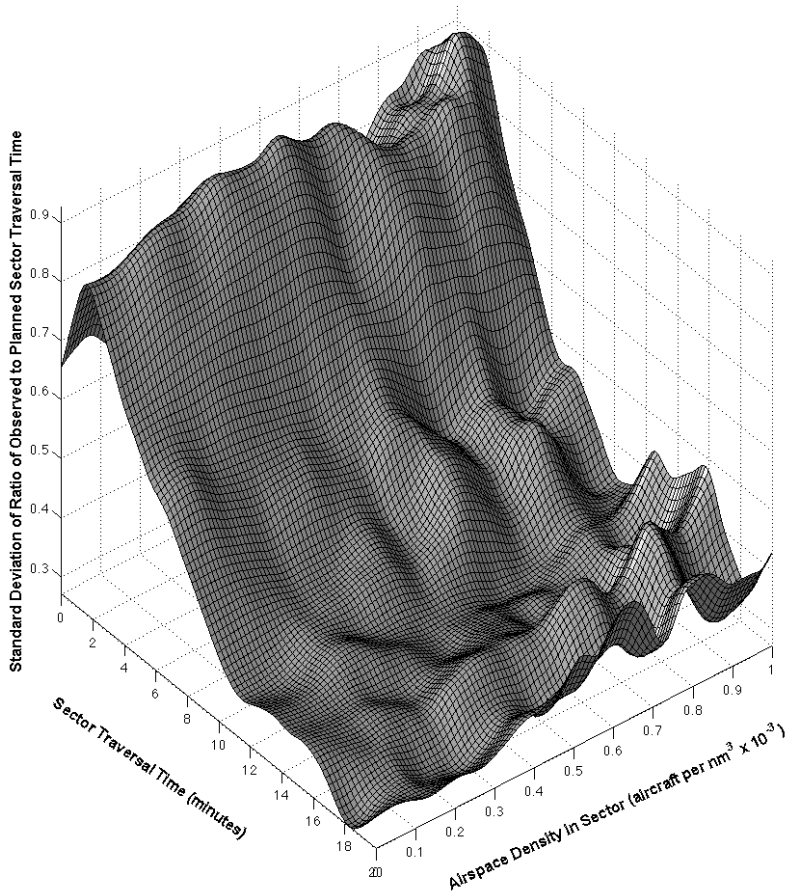

Figure 22: Standard Deviation of Ratio of Observed to Planned Sector Traversal Time by Planned Sector Traversal Time and Observed Airspace Density in the Sector.

Based on this observation a series of kernel-smoothed densities are developed for planned traversal times (t_p) as follows:

- $\{t_p | 0 < t_p \leq 4\, minutes\}$
- $\{t_p | 4 < t_p \leq 8\, minutes\}$
- $\{t_p | 8 < t_p \leq 12\, minutes\}$
- $\{t_p | 12 < t_p \leq 16\, minutes\}$
- $\{t_p | 16\, minutes < t_p\}$.

A sample kernel-smoothed ratio for the 4 to 8 minute planned traversal time interval is shown in Figure 23.

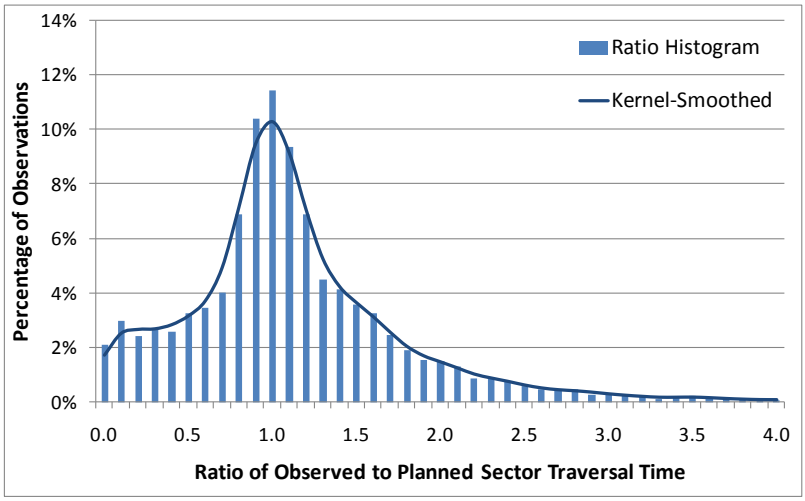

Figure 23: Ratio of Observed to Planned Sector Traversal Time for a Planned Traversal Time of 4 to 8 Minutes.

Alternatively, an error distribution that considers the relative difference between the observed and planned traversal times is also considered but not selected (i.e. error distribution = observed traversal time - planned traversal time). Due to the difference between the high and low range, e.g. 4 to 8 minutes in Figure 23, a negative sector traversal time may be implied from the resulting error distribution. The ratio distribution is more appropriate in this case since traversal times are always positive and relative to the planned traversal time.

5.3 Sector Hit Rate

The last source of uncertainty considered in this analysis is the sector hit rate which is defined as the rate at which the planned sectors for a flight plan match the observed or flown sequence of sectors. It is a combined consideration of the route and altitude forecast accuracy. Consider ordered sets of planned sectors P and flown sectors F. A hit is defined if the planned and flown sectors match and is then used to calculate the overall hit rate (Equation 30). Note that it is possible for a flight to enter the same sector more than once so by this definition the hit rate is restricted to be ≤ 1.

$$Hit\ Rate = \frac{|P \cap F|}{|P|} \tag{30}$$

The simulation and playback results from the sector traversal analysis in RAMS are also used to calculate the hit rate. The results of the hit rate analysis show an overall average hit rate of 73%. Conditions that lead to re-routing such as severe weather and airspace congestion were not included here. Further work that could find a relationship to predict sector hit rate probabilities under various conditions would be beneficial.

5.4 Probabilistic Sector Demand

We now use the departure uncertainty and sector traversal variability models to develop sector demand at 10, 30, and 60 minute look-ahead times to sector entry. MATLAB is used to perform all calculations listed in this section. The analysis scripts are included in Appendix B starting on page 192.

We are interested in the probability for a given flight to occupy a sector as a function of time (Figure 24). The resulting distribution is not a probability density function since the area under the curve does not equal 1. This distribution would then be used in the calculation of sector demand by time period.

The equations in this section represent the application of standard statistical methods, such as the convolution theorem (Champeney, 1987), and conventions used in calculations. The following nomenclature is used throughout this section:

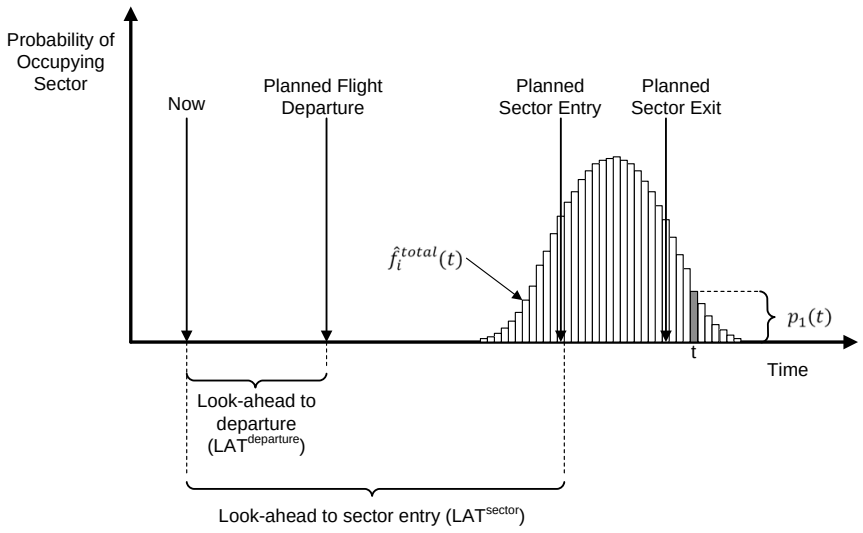

Figure 24: Distribution of Probabilistic Sector Demand.

$\hat{d}_i(t)$ = Expected count, or demand, for sector i, during time period t

$f_i^{departure}(\cdot)$ = Distribution of errors in predicting a flights departure time as calculated in Section 5.1 for the flight traversing the sector at position i under consideration

$f_i^{enroute}(\cdot)$ = Distribution of flight traversal time through sector at position i considering the stochastic en route component

$f_i^{total}(\cdot)$ = Probabilistic demand distribution for sector at position i considering both departure and en route sources of randomness

$f_k^{ratio}(\cdot)$ = Distribution of ratio r for sector k as calculated in Section 5.2

F = Set $\left\{s_1^{flown}, \ldots, s_m^{flown}\right\}$ of sectors traversed using the flight's flown trajectory

i, j = Positiong indices where position 1 is the first sector after the departing airport and positions m, n are the last sector before the arrival airport

k = Sector index for ratio distribution $f_k^{ratio}(\cdot)$

$LAT^{departure}$ = Look-ahead time to departure (wheels-off)

LAT^{sector} = Look-ahead time to sector entry

m = Number of observed sectors that a flight crosses when following the flown trajectory

n = Number of sectors that a flight crosses when following its flight plan

$p_0(t)$ = Probabilisty that the flight under consideration arrives to a sector i during time period t

$p_1(t)$ = Probabilisty that the flight under consideration does not arrive to a sector i during time period t

P = Set $\left\{\hat{s}_1^{planned}, \ldots, \hat{s}_m^{planned}\right\}$ of sectors in the flight plan

r = Ratio of observed $t_j^{traversal}$ to planned $\hat{t}_i^{traversal}$ traversal times

$\hat{s}_i^{planned}$ = Sector in position i in the flight plan

s_j^{flown} = Sector at position j in the set of flown sectors F

$\hat{t}_i^{entry}$ = Planned time of entry into a sector at position i

$\hat{t}_i^{LAT}$ = Planned time of entry into a sector at position i at look-ahead time LAT

$\hat{t}_i^{traversal}$ = Planned traversal time through a sector at position i at look-ahead time LAT

t_j^{entry} = Flown, or observed, time of entry into a sector at position j

$t_j^{traversal}$ = Flown, or observed, traversal time through a sector at position j

t_k^T = Variable used to convert from a ratio distribution to a relative error distribution

The analysis of historical ETMS data presented is sector based so to generate demand each sector in a flight plan is examined. To start we consider a single flight, its associated flight plan, and one of the sectors that the flight traverses when it follows its flight plan.

There are two cases to be considered for demand prediction for en route flights. There are additional considerations for flights that have not departed that are discussed later in this section. In the first case we find a flown sector that satisfies the conditions listed in Equations 31 to 33. The first of these conditions is that the flown sector must also be included in the set of planned sectors (Equation 31). As shown in Section IV there are cases where the flown sector does not appear in the flight plan. The second condition ensures that

the flown sector is at least the look-ahead time away from the planned sector (Equation 32). The third condition specifies that there is no closer flown sector (Equation 33).

So if a sector is found that satisfies the three conditions in Equations 31 to 33 an improved estimate of the estimated sector entry time is calculated (Equation 34). Otherwise, for the second case where no sector is found that satisfies Equations 31 to 33 the uncorrected planned time of entry into a sector is used which is the second condition in (Equation 34).

$$s_j^{flown} \in P \tag{31}$$

$$\hat{t}_i^{entry} - t_j^{entry} \geq LAT^{sector} \tag{32}$$

$$\{s_k^{flown} \in F \mid LAT^{sector} \leq \hat{t}_i^{entry} - t_k^{entry} < \hat{t}_i^{entry} - t_j^{entry}\} = \emptyset \tag{33}$$

$$\hat{t}_i^{LAT} = \begin{cases} \hat{t}_i^{entry} + \left(t_j^{entry} - \hat{t}_j^{entry}\right), & if\ \exists\ s_j^{flown} \\ \hat{t}_i^{entry}, & otherwise \end{cases} \tag{34}$$

The next step is to determine the set of sectors for which traversal time ratio distributions will be considered and included in the analysis. If a flown sector is found that satisfies Equations 31 to 33 then all sectors after and including the flown sector are included, otherwise all sectors are used to update the uncertainty distribution starting from the origin airport (Equation 35). Each of these sectors is matched with an appropriate ratio distribution that is described in Section 5.2 and categorized by the ratio of observed to planned sector traversal times (Equation 36). Since we are interested in the time relative to the corrected sector entry time calculated in Equation 34 we convert the basis of the distribution in Equation 37. Planned traversal times are subtracted for all sectors excluding the planned sector under consideration so that all distributions are error distributions except the planned sector under consideration. For the planned sector under consideration the expected traversal time is included in the distribution to achieve a correct demand value.

$$j = \begin{cases} j, & if\ \exists s_j^{flown} \\ 1, & otherwise \end{cases} \tag{35}$$

$$f_k^{ratio}(r; \hat{t}_k^{traversal}), \forall k = j, \ldots, i \tag{36}$$

$$t_k^T = \begin{cases} (r-1)\hat{t}_k^{traversal}, & if\ j \leq k < i \\ (r)\hat{(t)}_k^{traversal}, & if\ k = i \end{cases} \quad \forall k = j, \ldots, i \tag{37}$$

The distributions are summed by the standard convolution (i.e. the $*$ operator) method of taking the discrete Fast Fourier Transform $\mathcal{F}$ of each of the distributions, performing an element-by-element multiplication, then transforming back using the Inverse Fast Fourier Transform $\mathcal{F}^{-1}$ (Champeney, 1987). The general case is shown in Equation 38 for distributions $f(\cdot)$ and $g(\cdot)$ and in Equation 40 for the distributions considered here. If a sector is found satisfying Equations 31 to 33 then the result of Equation 40 is the density function relative to the corrected sector entry time to be added to the sector demand. Otherwise the departure time prediction error is also considered.

$$(f * g) = \mathcal{F}^{-1}\left[\mathcal{F}(f) \cdot \mathcal{F}(g)\right] \tag{38}$$

$$f_i^{enroute}(t) = \left[\left(f_j^{ratio}(t_j^T) * f_{j+1}^{ratio}(t_{j+1}^T)\right) \cdots f_i^{ratio}(t_i^T)\right] \tag{39}$$

$$= \mathcal{F}^{-1}\left[\prod_{k=j}^{i} \mathcal{F}\left(f_k^{ratio}(t_k^T)\right)\right] \tag{40}$$

If the departure time prediction error needs to be considered then the look-ahead time for the departure uncertainty is calculated using the look-ahead time to the sector, the planned entry time into the sector, and the planned flight departure (Equation 41). The departure look-ahead time is rounded up to one of the available departure look-ahead times of 0, 15, 30, 60, 120 minutes and used to select a departure uncertainty distribution. The departure uncertainty distribution is combined with the en route uncertainty distribution to arrive at a total uncertainty distribution (Equation 42).

$$LAT^{departure} = LAT^{sector} - \left(\hat{t}_i^{entry} - \hat{t}^{departure}\right) \tag{41}$$

$$f_i^{total}(t) = \left[f_i^{departure}(t) * f_i^{enroute}(t)\right] \tag{42}$$

To estimate the demand for the planned sector under consideration during any time period a summation of the discrete total error distributions is performed (Equation 43). If a distribution of demand for a sector is required rather than just the expected count then a discrete probability density function is constructed for each flight consisting of the probability that the flight arrives during a time period p_1 (Equation 44) or does not arrive p_0 (Equation 45). A series of convolution operators for each flight similar to that shown in Equation 40 may be used to derive a distribution of sector counts for the purpose of obtaining confidence bounds.

$$\hat{d}_i(t) = \sum_{\forall\, flights} f_i^{total}(t) \tag{43}$$

$$p_1(t) = P(N=1) = f_i^{total}(t) \tag{44}$$

$$p_0(t) = P(N=0) = 1 - p_1 \tag{45}$$

The method presented in this section implicitly assumes statistical independence for the departure and en route error distributions. Though this assertion is not strictly true it does allow for efficient demand uncertainty calculations. Methods that consider the covariance between the sector-based uncertainty distributions would also need to be computationally efficient to be useful for strategic traffic flow management.

5.5 Performance of Probabilistic Sector Demand Model

The historical traffic conditions on the date of August 29, 2005 is used to compare the performance of the probabilistic model for sector demand to a deterministic model at 10, 30, and 60 minute look-ahead times to sector entry in 1-minute intervals. Recall from the departure uncertainty section that two groupings are considered: one based on departing airport type and one based on a clustering that considers additional factors. Overall comparisons are made by considering the standard deviation of the demand prediction error and the mean absolute value of the prediction error (Table 21). The standard deviation of the error is reduced by 25% and the prediction error reduced by 20% when using the probabilistic methods as compared to the deterministic method. Results indicate that the cluster grouping

method that considers additional factors offers little improvement on the method that only considers airport type in the departure uncertainty. Both methods also consider the en route random component as described in Equation 40. A histogram detailing the distribution of the prediction error at a 30 minute look-ahead time to sector entry is shown in Figure 25. This deterministic to probabilistic comparison is challenged by the fact that deterministic errors are discrete whereas the probabilistic errors may take on any real value.

Table 21: Comparison of Deterministic and Probabilistic Methods.

	Deterministic Demand	Probabilistic Demand (Airport Grouping)	Probabilistic Demand (Cluster Grouping)
S.D.[a] 10 min.[c]	1.909	1.487	1.485
S.D.[a] 30 min.[c]	1.971	1.499	1.497
S.D.[a] 60 min.[c]	2.003	1.508	1.507
M.A.P.E.[b] 10 min.[c]	1.109	0.900	0.899
M.A.P.E.[b] 30 min.[c]	1.144	0.909	0.907
M.A.P.E.[b] 60 min.[c]	1.159	0.914	0.913

[a] Standard deviation of the prediction error.

[b] Mean absolute value of the prediction error.

[c] Look-ahead time in minutes.

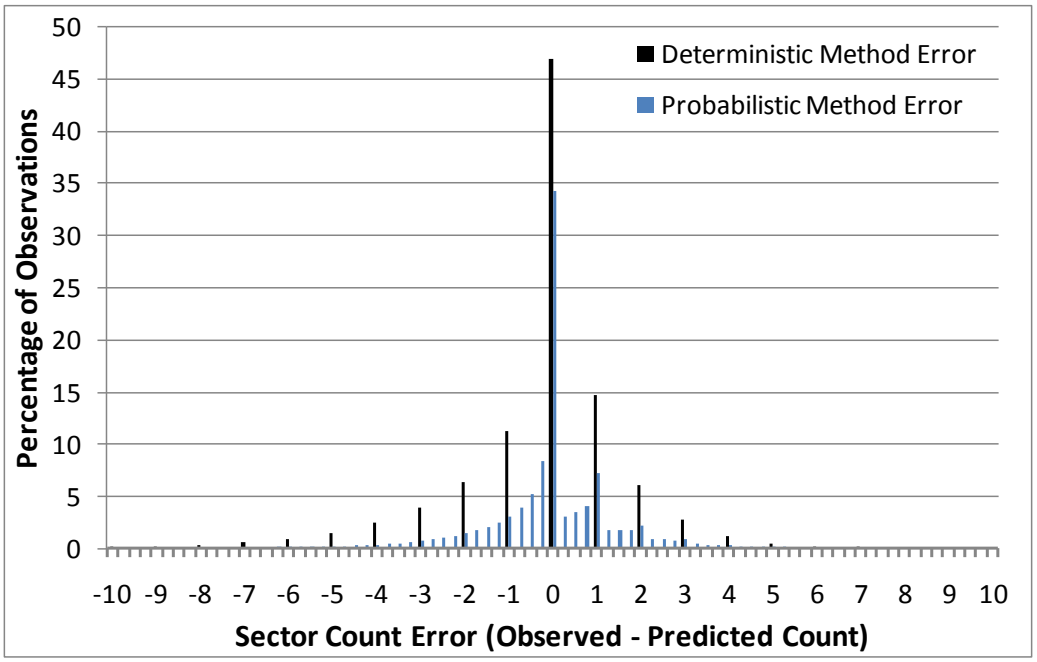

Figure 25: Histogram Comparison of the Distribution of Sector Count Errors at a 30 Minute Look-Ahead Time for Deterministic and Probabilistic (Airport Grouping) Methods.

6 STOCHASTIC AIRSPACE CAPACITY

This section describes the process of estimating en route sector capacity considering uncertain convective weather and flight demand information. Analysis focuses on applying stochastic geometry techniques to create random separation and conflict potential models.

6.1 Probabilistic Weather Forecasts

To estimate the impact on en route airspace of a convective weather system the distribution of location and extent of the convective weather must be known. The aviation weather forecasts described in the literature review (Section 2.4) are not directly able to provide this information. The deterministic forecasts produce an expected location and extent but not a distribution of such quantities. The experimental probabilistic forecasts are cell-based and cannot be used to develop the required spatial relationships. For example, given a probability of convective weather in cell A we are unable to derive the probability of convective weather also occurring in cell B. Due to this shortcoming the probabilistic forecasts cannot provide a distribution of the extent of the forecasted convective weather system.

To overcome the shortcomings of existing convective weather forecasts we compare historical National Convective Weather 1-hour Forecasts (NCWF) to the observed convection using the National Convective Weather Detection (NCWD) product. NCWD contains a model to combine lightning strikes within 8 km of a grid point and the vertically integrated liquid (VIL) mosaic data to produce a 2-D convective intensity estimate on a 4-km grid (Megenhardt et al., 2004). NCWF and NCWD image data are extracted from NOAA's Real-Time Verification System (Mahoney et al., 2002) for June and July 2006 at 1-hour intervals.

The first step in the image processing is to add a buffer around the NCWD cells (Figure 26). The MATLAB script used in the image processing is included in Appendix C starting on page 201. The buffer is required since flights must avoid turbulence in the area of the convective activity (FAA, 1983). Using a specified buffer the bias in the observed forecast is calculated in Equation 46. This bias varies by the area of the forecast due to the threshold for reporting forecasted convection (Figure 27). To account for this thresh-

old a series of Weibull distributions (Equation 47) are fitted by forecast area increments of 50 nm^2 (Table 22). Centroid-to-centroid distance, defined as the distance between the NCWD observed convection and the closest NCWF forecasted convection, is also included in Table 22.

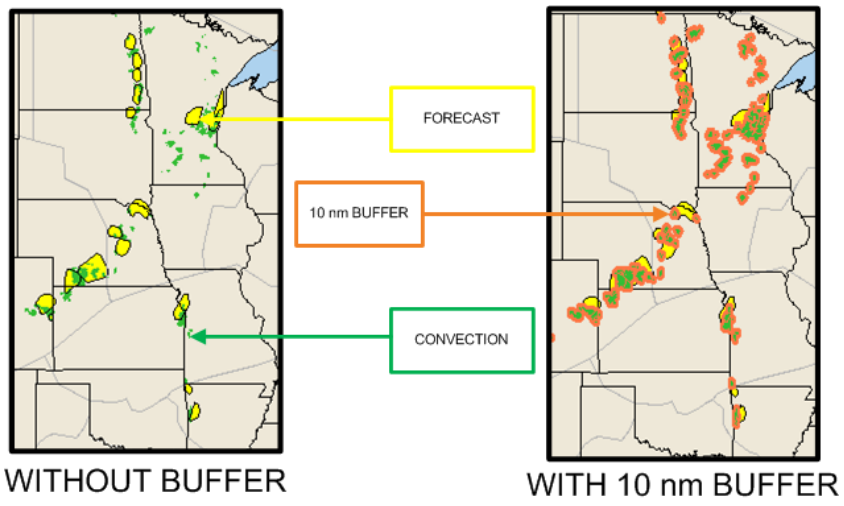

Figure 26: Addition of a 10 nm Buffer to the National Convective Weather Detection (NCWD) Product (Background Image: NOAA Earth System Research Laboratory (2007).

$$Bias = \frac{NCWD\ Convection\ Area + Buffer\ Area}{NCWF\ Forecast\ Area} \tag{46}$$

$$f(x;k,\lambda) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-\left(\frac{x}{\lambda}\right)^{k}} \tag{47}$$

Random variates are drawn from the Weibull distribution and the corresponding size inflation/deflation and translation transformations are applied to simulate the convective weather systems. All directions of translation are assumed to have equal probability so the generation of an azimuth is drawn from a uniform distribution of 0 to 360 degrees. The intersecting area of the forecasted convection with airspace sectors is calculated using the MATLAB mapping toolbox. This intersecting area is considered the reduction in available airspace volume when combined with either the sector height or the tops of the convective weather system.

Table 22: Fitted Weibull Parameters for Distributions of Bias and Centroid-to-Centroid Distance.

	Bias for 0 nm Buffer		Bias for 10 nm Buffer		Bias for 20 nm Buffer		Centroid-to-Centroid Distance	
Forecast Area (nm^2)	λ	k	λ	k	λ	k	λ	k
0	1.35	0.83	9.91	0.88	26.07	0.87	81.76	0.47
50	1.75	0.83	11.35	0.86	29.09	0.84	75.03	0.51
100	3.18	0.89	16.06	0.81	39.02	0.77	61.51	0.87
150	4.16	0.83	20.05	0.72	48.44	0.68	83.86	0.86
200	4.73	0.81	21.74	0.68	51.88	0.64	94.15	0.84
250	5.23	0.77	23.65	0.64	56.33	0.60	110.20	0.87
300	6.64	0.75	31.25	0.62	76.29	0.58	146.30	0.92
350	7.02	0.72	32.41	0.57	80.08	0.54	129.13	0.89
400	5.59	0.77	25.01	0.60	59.93	0.57	140.39	1.00
450	7.24	0.81	35.15	0.64	84.41	0.61	165.44	0.92
500	12.04	0.69	63.61	0.56	157.88	0.54	207.76	0.92
550	12.28	0.61	66.58	0.51	166.61	0.49	244.29	1.29
600	14.05	0.65	73.18	0.54	183.38	0.52	314.37	1.21
650	17.70	0.73	92.98	0.59	236.04	0.57	297.59	1.14
700	7.31	1.13	34.18	0.80	85.71	0.74	196.39	1.31

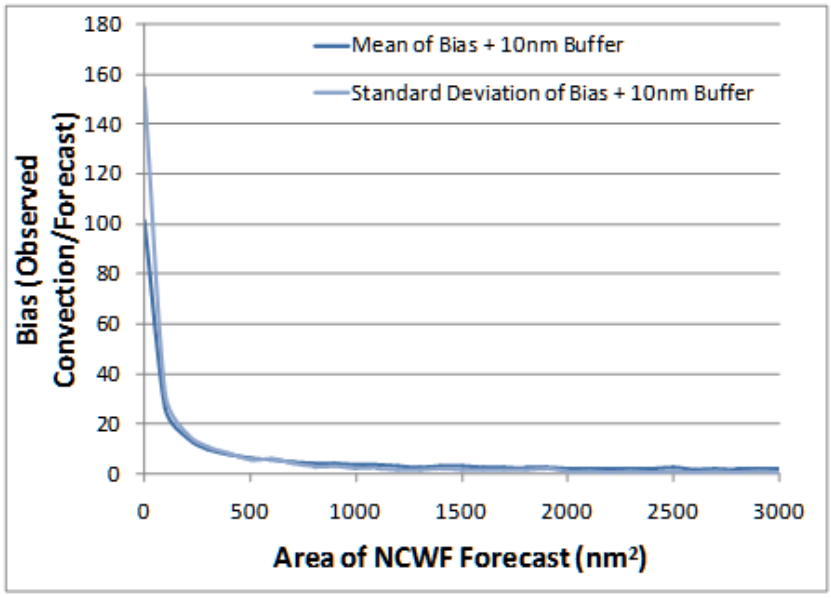

Figure 27: Bias of Forecast Area for a 10 nm Buffer.

Next we consider the height of the weather system to extend the 2-D analysis presented above to the 3-D case. The Collaborative Convective 1-2 hour Forecast Product (CCFP) is also monitored by the Real-Time Verification System so that an estimate of the historical bias of the system may be calculated. Table 23 shows that the weather forecast is usually conservative and that the probability of exceeding the forecast is low. A standard roulette-wheel type generation is used to simulate the height of the weather system during scenario generation (Law, 2007).

Table 23: Performance of CCFP Forecasted Convection Tops.

		Observed Convection Tops (1000's feet)				
		N/A	< 25	25-31	31-37	> 37
Forecasted	25-31	3%	75%	21%	1%	0%
Convection Tops	31-37	2%	39%	58%	2%	0%
(1000's feet)	> 37	1%	20%	68%	11%	1%

6.2 Separation Distribution Prediction

We now estimate the impact a reduction in available airspace has on capacity by considering the metrics of aircraft separation and the expected rate of conflict resolution actions by the en route air traffic controller. Separation between aircraft is a key indicator of collision risk and controller workload and is modeled as a function of: flights occupying the airspace, area or volume of airspace available, aircraft speed, traversal distance through the sector, and the structure of the jet routes.

Prior work in this area has focused on detailed trajectory modeling with probabilistic deviations to predict separation and collision risk (Sherali et al., 2000). This approach has some drawbacks in the context of en route airspace planning. Detailed trajectory modeling for large-scale problem instances requires extensive computational resources and the time required for calculations may exceed the planning horizon to the anticipated weather or capacity constrained event. Another issue is the accuracy of the planned routes. As discussed in Section 5.3 it is common that the flown trajectory of the aircraft is different from the planned route. So detailed modeling of an uncertain quantity is unlikely to produce higher quality results when compared to an approximate method. To overcome these shortcomings we propose to use stochastic geometry techniques to develop a random aircraft separation model.

Stochastic geometry is the field studying geometric objects with random properties such as: location, intensity, interaction with other geometric objects, and markers or labels. A rigorous treatment of stochastic geometry is included in any of (Stoyan et al., 1987; Okabe et al., 2000; Moller and Waagepetersen, 2004), which is excluded here since we are only interested in the summary properties of spatial Poisson point processes. In Poisson point processes the points, which are centers of objects in 2 or higher dimensions, are uniformly distributed throughout the defined set/area/volume. For Poisson point processes the count of points in a subset area or volume follows a Poisson distribution. The derivation of first contact separation (Figure 28) is presented in several places (e.g. Stoyan et al. (1987)) and a simplified version is presented for the sake of completeness.

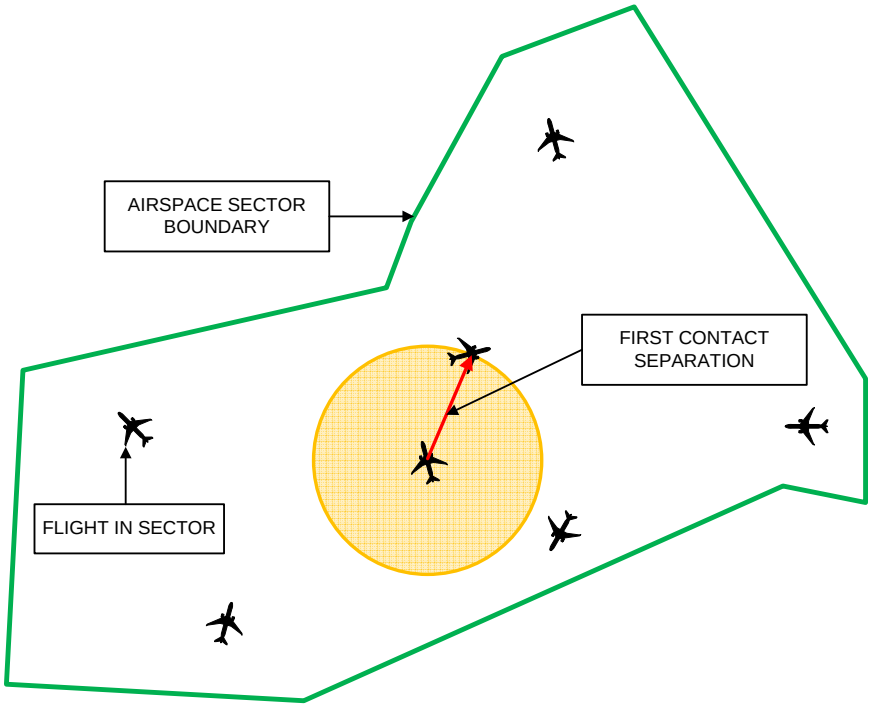

Figure 28: First Contact Separation for Flights in an Airspace Sector.

The following nomenclature is used in the derivation of first contact aircraft separation:

δ = Correction factor to account for the jet route structure

λ_0 = Intensity of the Poisson process as mean number of flights per unit of airspace ($\frac{flights}{nm^2}$)

μ = Model estimated mean separation between aircraft (nm)

μ_δ = Observed mean separation between aircraft (nm)

$E(\cdot)$ = Expected value

$f(\cdot)$ = Probability density function (pdf)

$F(\cdot)$ = Cumulative density function (cdf)

N = Aircraft count

P = Probability of

r = Radius distance around the aircraft used to specify a minimum separation shell (nm)

We begin the derivation by specifying an intensity for the point process (Equation 48). The numerator of the equation is the expected count of flights in the airspace sector which

is divided by the area of the sector to get the expected number of flights occupying an area of airspace. Intensity increases as the arrival rate or distance traveled through the sector increases. Alternatively, intensity decreases as sector area increases or aircraft speed is increased.

$$\lambda_0 = \frac{(Arrival\ Rate\ to\ Sector)\,(Traversal\ Time\ Through\ Sector)}{Sector\ Area} \tag{48}$$

If the distribution of flights within an area of airspace follows a Poisson distribution then the probability of no flights occupying a circle surrounding the flight is the same as the probability of the Poisson count being zero (Equation 49). Analysis that follows the derivation of minimum contact separation confirms that this is a good approximation.

$$P\left(N = 0 | R = r\right) = \frac{e^{-\lambda_0 r^2 \pi} \lambda_0^0}{0!} = e^{-\lambda_0 r^2 \pi} \tag{49}$$

The cumulative distribution of the first contact distance is obtained by noting that the complement of the probability that no flights are within a radius is the probability of one or more flights within the radius (Equation 50).

$$F\left(\lambda_0, r\right) = 1 - P\left(N = 0 | R = r\right) = 1 - e^{-\lambda_0 r^2 \pi} \tag{50}$$

Taking the derivative of the cumulative distribution results in the probability density function shown in Equation 51. This is the Weibull (or equivalently Rayleigh) distribution (Equation 47) with parameters shown in Equations 52 and 53. The mean separation of a Weibull distributed random variable is shown in Equation 54.

$$f\left(\lambda_0, r\right) = \frac{dF\left(\lambda_0, r\right)}{dr} = 2\pi r \lambda_0 e^{-\pi r^2 \lambda_0} \tag{51}$$

$$\lambda = \frac{1}{\sqrt{\pi \lambda_0}} \tag{52}$$

$$k = 2 \tag{53}$$

$$\mu = \frac{1}{2\sqrt{\lambda_0}} \tag{54}$$

This model assumes that flights are randomly distributed throughout the available airspace. This is not strictly the case since flight plans generally follow the airway and jet route structure of the NAS. However, analysis of radar trajectories on August 29, 2005 for flights operating on the east coast of the U.S. (Chicago (ZAU), Indianapolis (ZID), Atlanta (ZTL), Jacksonville (ZJX), Miami (ZMA), Washington (ZDC), Cleveland (ZOB), New York (ZNY), and Boston (ZBW)) indicates that this is a good approximation (Figure 29). The predicted average separation for the model is 65 nm, which is close to the observed separation of 64 nm. For this analysis flights operating at flight level 350 (35,000 feet) are extracted at 1 minute intervals and processed in MATLAB using the mapping toolbox to obtain the closest flight (Appendix C, page 214). The intensity levels are observed 15 minute sector averages so the quality of the forecasted sector counts are not considered.

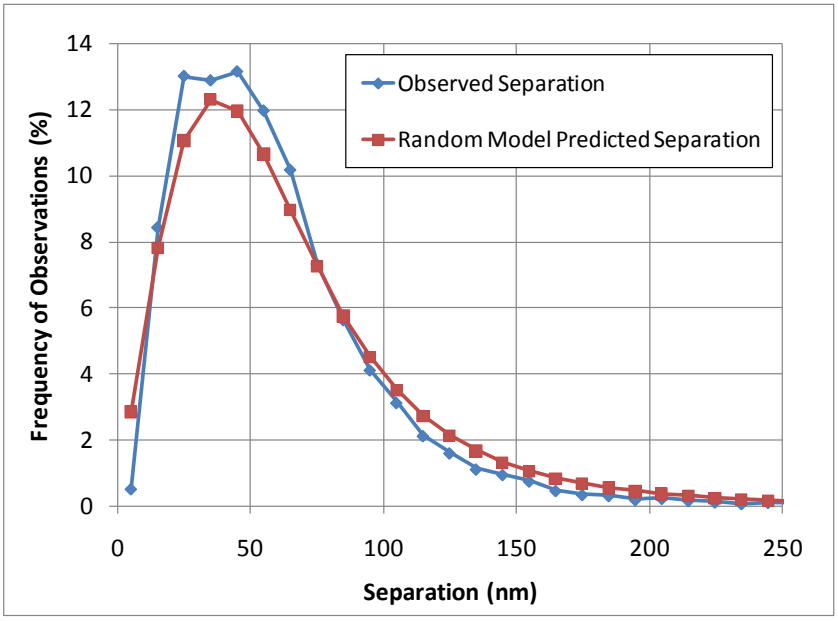

Figure 29: Distribution of Minimum Separation (First Contact Distance) between Flights operating at Flight Level 350 on August 29, 2005 in the Eastern U.S.

To account for the jet route structure we propose a correction factor to the intensity estimate (Equations 55 and 56). The correction factor is obtained by substituting Equation 55 into Equation 54 and rearranging to obtain the correction factor as a ratio of the squared model estimated mean separation to the squared observed separation (Equation 57). Flight

radar tracks are extracted from ETMS data at 1 minute intervals for flights operating above 18,000 feet on August 29, 2005 to obtain the correction factor . The 15 minute demand for a sector is estimated by assigning each flight position to a sector. The average intensity over the 15 minute time period is then used to estimate the first contact separation distribution of the flights. The model estimated first contact separation is compared to the observed separation in 15 minute interval bins to obtain the correction factors shown in Table 24. Table 24 includes flights at all altitudes in the sector and the results are based on a radar scope plan view separation.

$$\lambda_\delta = \delta\lambda_0 \tag{55}$$

$$f_\delta(\lambda_\delta, r) = 2\pi r\lambda_\delta e^{-\pi r^2\lambda_\delta} \tag{56}$$

$$\delta = \frac{\mu^2}{\mu_\delta^2} \tag{57}$$

The correction factor is reasonable in most cases, 12 out of 18 have correction factors between 0.75 and 1.25, however there are some sectors where the separation model has lower predictive quality. Based on the sector-centric results shown in Table 24 and the results at one altitude band across several sectors shown in Figure 29 it is clear that this model is more applicable on a larger scale. The observed general trend is that as the model is applied to scenarios of smaller geographic extent the quality of separation prediction declines.

If it could be demonstrated that the sector-based correction factor is stable from day to day then the random separation model would still be useful. The correction factors obtained on the additional date of July 27, 2005 is used as a comparison to the factors calibrated for August 29, 2005. As seen in Table 25 even though the predicted and observed first contact separations change the correction factors remain stable for most sectors.

6.3 Expected Number of Conflict Resolution Actions

We estimate the en route controller workload associated with conflict resolution actions by extending the model presented in the previous section. This model is more appropriate for strategic air traffic flow management than current conflict models that must consider every flight pair and potential conflict location in a sector (Figure 30). The goal is to have a

Table 24: Correction Factors for Washington (ZDC) Sectors Obtained by Processing Flight Radar Track Data on August 29, 2005.

Sector	Area of Sector (nm^2)	Sector Floor (1000's feet)	Sector Ceiling (1000's feet)	Observed Average First Contact Separation (nm)	Predicted Average First Contact Separation (nm)	Observations (aircraft-minutes)	Correction Factor
ZDC72	10,997	33.1	99.9	29.0	25.2	3,660	0.76
ZDC60	9,134	13.1	27.0	42.3	43.1	1,896	1.04
ZDC59	5,410	18.0	99.9	24.6	17.5	3,099	0.51
ZDC58	3,308	18.0	99.9	20.4	22.2	1,577	1.18
ZDC54	4,437	23.1	33.0	23.4	20.6	2,383	0.78
ZDC52	7,518	23.1	33.0	29.8	26.2	1,988	0.77
ZDC50	13,118	33.1	99.9	24.1	16.4	6,457	0.46
ZDC38	5,409	23.1	99.9	19.9	23.3	3,354	1.37
ZDC37	4,057	23.1	99.9	20.8	16.3	2,190	0.61
ZDC36	5,532	23.1	99.9	20.3	32.9	3,569	2.63
ZDC35	10,873	23.1	33.0	39.7	36.5	1,429	0.85
ZDC34	17,155	23.1	33.0	37.9	28.8	1,852	0.58
ZDC33	8,747	0.0	23.0	51.5	47.5	539	0.85
ZDC32	9,793	21.1	33.0	31.0	27.8	2,918	0.80
ZDC20	7,981	20.1	27.0	38.7	23.7	2,027	0.37
ZDC19	6,139	8.0	99.9	18.3	16.0	3,149	0.76
ZDC16	5,234	27.1	99.9	21.0	22.2	3,501	1.12
ZDC12	11,021	23.1	99.9	24.5	21.9	5,412	0.80

Table 25: Correction Factors for Washington (ZDC) Sectors Obtained by Processing Flight Radar Track Data on July 27, 2005 and Compared to Correction Factors Obtained for August 29, 2005.

Sector	Observed Average First Contact Separation (nm)	Predicted Average First Contact Separation (nm)	Observations (aircraft-minutes)	Correction Factor (July 27)	Correction Factor (August 29)
ZDC36	20.5	32.1	3,958	2.44	2.63
ZDC38	19.8	22.7	3,692	1.32	1.37
ZDC60	35.0	39.5	2,304	1.27	1.04
ZDC16	18.7	20.3	4,277	1.19	1.12
ZDC33	50.0	51.1	446	1.05	0.85
ZDC32	26.5	26.2	3,698	0.98	0.80
ZDC58	21.9	21.3	2,035	0.94	1.18
ZDC35	33.0	31.0	2,184	0.89	0.85
ZDC12	21.1	19.8	7,163	0.89	0.80
ZDC72	24.5	22.8	5,505	0.87	0.76
ZDC54	21.2	18.9	3,213	0.79	0.78
ZDC52	29.3	25.8	2,214	0.78	0.77
ZDC19	19.5	16.4	3,366	0.70	0.76
ZDC59	19.5	15.7	4,113	0.65	0.51
ZDC37	18.9	14.9	2,998	0.62	0.61
ZDC20	33.0	24.1	2,126	0.53	0.37
ZDC50	21.8	14.7	8,798	0.45	0.46
ZDC34	38.2	24.4	2,748	0.41	0.58

general idea of the rate of conflicts but it is acknowledged that this model does not contain sufficient detail to study the related problem of collision risk potential.

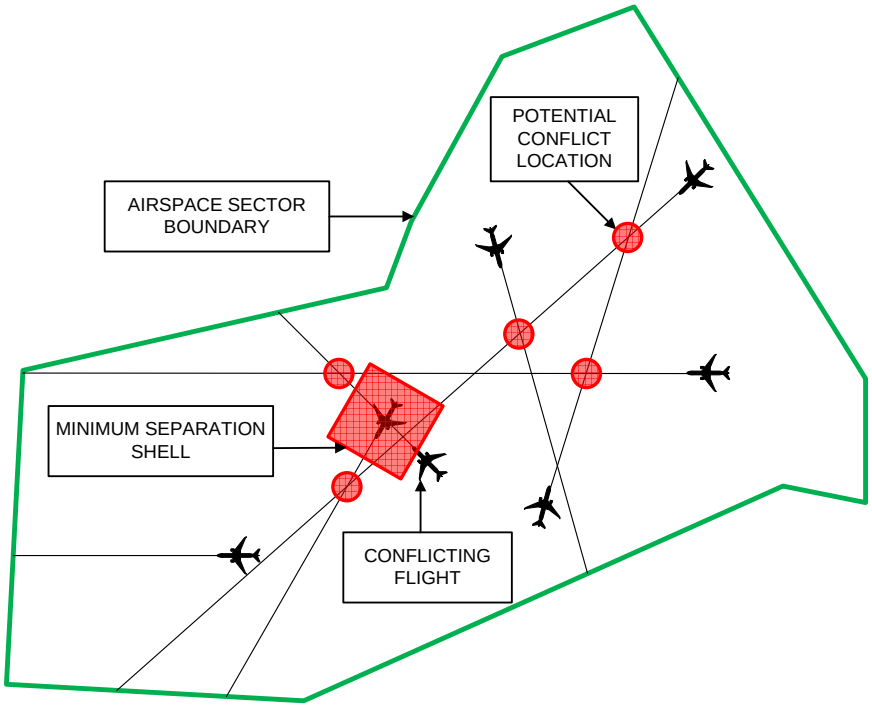

Figure 30: Shortcoming of Existing Conflict Models that Must Consider Every Pair of Conflicts and Conflict Locations.

The nomenclature of Section 6.2 is also used in this Section and is extended to include the following:

$$
\begin{aligned}
A_{sector} &= \text{Area of the sector } (nm^2) \\
N_{conflicts} &= \text{Number of conflicts} \\
N_{shell} &= \text{Number of flights within the minimum separation shell} \\
t_{conflict} &= \text{Time in conflict}
\end{aligned}
$$

Begin by noting that the Poisson distribution parameter is also the mean of the distribution (Equation 58). This is directly applicable to finding the expected number of aircraft that violate a minimum separation shell around a focal aircraft at an instant of time (Equation 59). More explicitly Equation 59 represents the intensity of flights in the sector multiplied by the

area of the minimum separation shell. To get the expected number of conflicts at a random instance of time multiply the probability of a conflict by the expected number of flights in the sector, which is the product of the intensity of flights in the sector and the area of the sector (Equation 60). Equation 60 also represents the proportion of time that there is a conflict present in the sector so multiplying by time will yield the time in conflict as shown in Equation 61. Applying the correction factor calibrated for the separation prediction model results in Equation 62.

$$\lambda = P(N=0)*0 + P(N=1)*1 + P(N=2)*2 + \ldots \quad (58)$$

$$P(conflict)E(N_{shell}) = \lambda_0 \pi r^2 \quad (59)$$

$$E(N_{conflicts}) = P(conflict)E(N)E(N_{shell}) = \left(\lambda_0 \pi r^2\right) \lambda_0 A_{sector} \quad (60)$$

$$E(t_{conflict}) = E(N_{conflicts})dt = \left(\lambda_0 \pi r^2\right) \lambda_0 A_{sector} dt \quad (61)$$

$$E(t_{conflict,\delta}) = \left(\lambda_\delta \pi r^2\right) \lambda_\delta A_{sector} dt \quad (62)$$

Preliminary validation of the conflict model is used to check for conceptual errors during model development. The New York center is the location for the MATLAB simulation of flights cruising at a constant azimuth, 35,000 feet (flight level 350), and a ground speed of 450 knots (Appendix C, page 233). The assumptions in this simulation match the assumptions for the random conflict model.

The simulation is run for a total of 500 flights at several inter arrival times to the center. Results of the simulation shown in Figure 31 indicate that the model and simulation produce similar results. Replications of each interarrival time scenario are not performed since the fit between model and simulation is acceptable.

For the analysis presented above to be useful the time in conflict should be converted to the expected number of conflict resolution maneuvers by dividing by the time per conflict. The time per conflict could potentially be derived analytically though this would be problematic due to assumptions regarding the angle of conflict and relative speed between aircraft. Figure 32 illustrates the wide range of aircraft speeds that can be expected, especially at lower altitudes. For these reasons a simulation method is used instead to arrive at the average time per conflict.

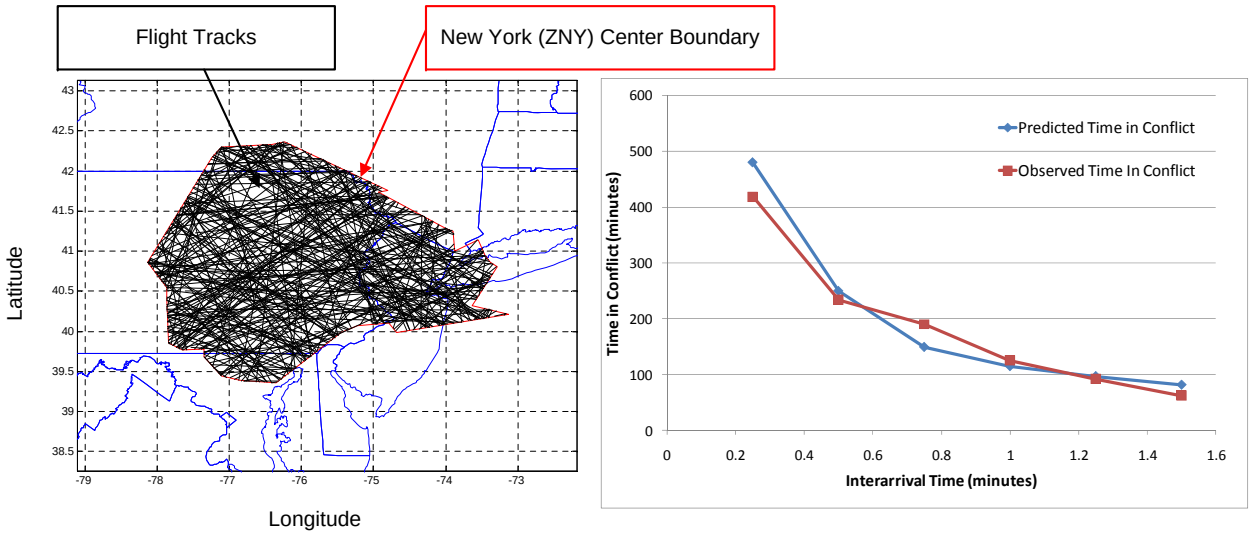

Figure 31: Screenshot of the MATLAB Simulation for Time in Conflict (Left) and the Corresponding Comparison of Model Predicted and Simulated (Observed) Time in Conflict for Various Interarrival Times (Right). The Screenshot is Based on an Interarrival Time of 0.7 Minutes.

The RAMS airspace simulator is used to process historical flight plans from June 11, 2004. The simulation uses blind conflict conditions so that controllers detect flights in conflict but do not issue conflict resolution maneuvers. A total of 13,372 blind conflicts are extracted to calculate the summary time per conflict shown in Table 27.

Table 27: Average Time per Conflict derived from Simulating Flight Plans from June 11, 2004.

Flight Level Lower (100's feet)	Flight Level Upper (100's feet)	Sample Size	Mean Time per Conflict (sec)
0	180	490	65.5
181	240	1,638	48.6
241	300	3,095	53.2
301	360	7,267	55.2
361	999	882	40.7

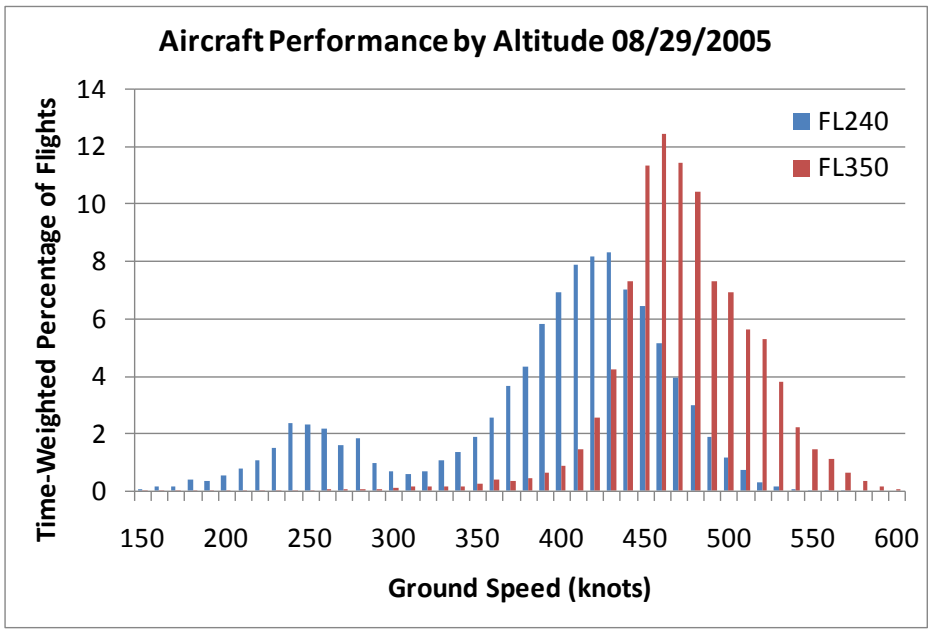

Figure 32: Aircraft Groundspeed Performance at Flight Levels 240 (24,000 feet) and 350 (35,000 feet) Showing Higher Variance at the Lower Altitude.

6.4 Controller Workload, Dynamic Density, and Sector Capacity

One method to apply the random separation and conflict prediction models is to use a defined sector capacity, such as the Monitor Alert Parameter (MAP) value, as the basis for changes to sector capacity. The MAP value could be used to establish an acceptable minimum separation and maximum conflict potential. For example, if a convective weather system reduces the available airspace then the random separation model could establish the maximum demand for the sector based on the acceptable minimum separation. A similar analysis could be performed based on conflict potential. The percentage decrease in available airspace does not necessarily correspond to the same percentage decrease in sector capacity due to potential differences in aircraft performance and traversal distance through the sector.

More advanced methods would consider the workload imposed on the controller and incorporate aspects of dynamic density. These topics are included in the literature review section (Section 2.10). Controller workload models could be used to account for two important factors: the reduction in workload associated with decreased intra-sector controller

handoffs and the increased workload associated with convective weather systems. Convective weather systems impose additional workload on the controller in the form of additional monitoring and pilot weather avoidance advisories. Unfortunately, appropriate controller workload models that consider severe weather in the en route airspace have not been developed and validated.

Instead we demonstrate how the random separation model can be applied to existing controller workload models. Consider the NASA vertical separation complexity metric C6 (Chatterji and Sridhar, 2001) that uses the following nomenclature:

d_{ij} = Horizontal separation between aircraft i and j (nm)

h_{ij} = Vertical separation between aircraft i and j (nm)

S_h = Ratio of minimum horizontal separation to minimum vertical separation. Based on current separation standards the value for this parameter is 5/1000.

The distribution of the C6 complexity metric cannot be derived analytically from the distributions of vertical and horizontal separation due to the form of the equations (Equations 63 and 64). Alternatively, it would be straightforward and computationally efficient to randomly generate horizontal separations at each of the flight levels in the sector and use these randomly generated separations to calculate C6. This type of simulation would be faster than detailed trajectory modeling and could be more appropriate when the trajectory uncertainty is high.

$$W_{ij} = \frac{[j \neq i]}{d_{ij}^2 + S_h^2 h_{ij}^2 + [j = i]} \tag{63}$$

$$C_6 = \frac{N}{\sum_{1 \leq i \leq N} \left(\frac{\sum_{1 \leq j \leq N} W_{ij} h_{ij}}{\sum_{1 \leq j \leq N} W_{ij}} \right)} \tag{64}$$

In this dissertation the focus is on the first and simpler method with the acknowledgement that when more appropriate controller workload models for severe weather become available the random separation and conflict prediction models could be further adapted.

6.5 Second Order Congestion Effects

As en route airspace sectors become more congested there is a potential for second order effects in the form of a reduction in airspeed or increased sector traversal time. Faster aircraft in-trail behind slower aircraft on a jet route may not be issued a vector around due to the high workload imposed on the controller or the potential for conflicts. Also, more direct routing to decrease sector traversal time may not be possible due to workload and conflict resolution issues.

To study this issue the flights trajectories extracted for the purpose of validating the random separation distribution model described in Section 6.2 are augmented with groundspeed values. The groundspeed observations are corrected for wind using standard vector operations to obtain an estimate of the airspeed. The airspeed estimate can then be compared to an aircraft performance model, such as BADA (Eurocontrol, 2003), to check for a reduction in speed. There is some uncertainty in aircraft performance modeling since the weight of the aircraft is not transmitted through the FAA's Enhanced Traffic Management System (ETMS). The assumed weight used is the nominal value reported for BADA.

Figure 33 shows that there does not seem to be a consistent relationship between separation and airspeed. Aircraft performance is a good indicator if the correct aircraft weight is selected. For the flights extracted at 35,000 feet shown on the right in Figure 33 an overestimation of the aircraft weight is made resulting in an under prediction of aircraft speed.

The results shown in Figure 33 seem to indicate that there is an inconsistent relationship between separation and airspeed. A similar analysis to estimate the relationship between separation and sector traversal time results in the same conclusion of an inconsistent relationship. It is possible that the airspace has not reached a level of congestion to achieve a stable relationship. Traffic flow managers and air traffic controllers are able to avoid highly congested conditions at present. In the future when there are more flights operating in the airspace this relationship may become clear but for the modeling purposes presented in this dissertation sector order congestion effects are ignored.

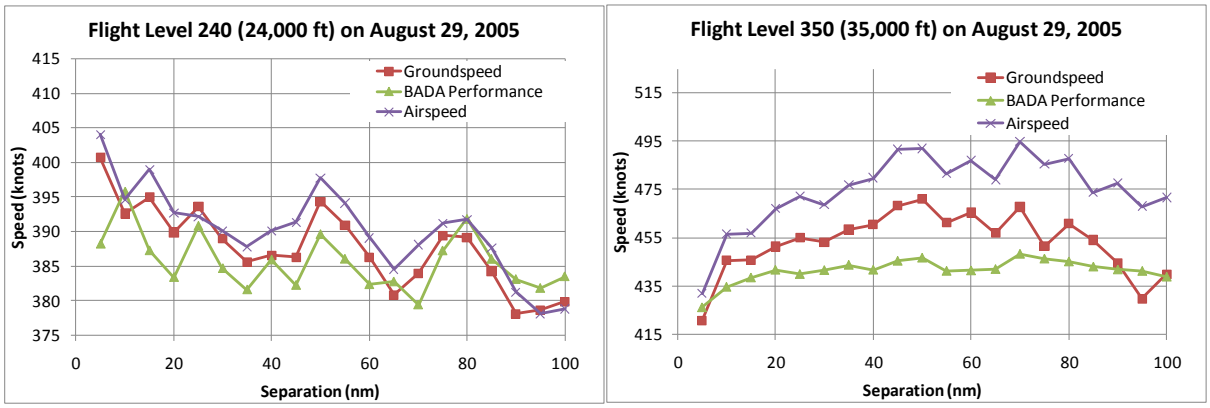

Figure 33: Aircraft Airspeed by Separation at 24,000 feet (Left) and 35,000 feet (Right).

7 ROUTING

This section describes the supporting models used to create efficient routing procedures during severe weather. First, the identification of congested airspace as a flow constrained area is discussed. This is followed by a procedure to group flights using clustering in an effort to reduce computational time. Lastly, the routing of each flight group is presented. Incremental assignment and an integer programming method are presented as alternatives to generate routes.

7.1 Flow Constrained Area

In this work we apply the simpler version of the method discussed in Section 6.4 that converts a reduction in available airspace to sector capacity. Demand and capacity are both evaluated at 15 minute intervals to estimate a flow constrained area (FCA). The FCA is used as a basis of selecting candidate flights for re-routing in an effort to reduce the problem size. The following nomenclature is used in the development of a flow constrained area:

λ_0	=	Baseline intensity at capacity ($\frac{aircraft}{nm^2}$)
$A_{effective}$	=	Effective area of the sector available to flights (nm^2)
A_{sector}	=	Area of the sector in plan view (nm^2)
$A_{weather}$	=	Area of the convective weather system (nm^2)
MAP	=	Monitor Alert Parameter (aircraft)
N	=	Sector count (aircraft)
$N_{capacity}$	=	Sector count at capacity (aircraft)
$N_{conflicts}$	=	Count of conflicts
p_{max}	=	Maximum probability for a sector to exceed capacity during a time period
t	=	Time index
$t_{\%,max}$	=	Maximum percentage of time that the probability of a sector exceeding its capacity is greater than p_{max}

The starting basis for this analysis is the FAA's Monitor Alert Parameter (MAP) value

that specifies the simultaneous number of aircraft in the sector (Equation 65). This value is not capacity in the strictest sense since it does not include other factors that can lead to increased workload for the controller but it is used here as a good approximation. A baseline intensity value is then established using the sector count at capacity (Equation 66).

$$N_{capacity} = MAP \tag{65}$$

$$\lambda_0 = \frac{N_{capacity}}{A_{sector}} \tag{66}$$

Recall that in this modeling framework several weather scenarios are to be generated (Section 4). For each of these weather scenarios the area of the sector that is unavailable is subtracted from the total area to get an effective area available for flights (Equation 67). The effective area is updated for each 15 minute analysis time interval and applied in the calculation of dynamic maximum sector occupancy (Equation 68). This is compared to the dynamic demand for sector resources (Equation 69) based on the principle that mean separation between aircraft should not decrease over the baseline conditions.

$$A_{effective}(t) = A_{sector} - A_{sector} \cap A_{weather}(t) \tag{67}$$

$$N_{effective}(t) = \lambda_0 A_{effective}(t) \tag{68}$$

$$N(t) = [arrival\, rate(t)]\,[traversal\, time(t)] \tag{69}$$

In this analysis separation between aircraft is checked so that it does not decrease to an unacceptable level. Using the minimum separation between aircraft is more critical than the conflict rate as demonstrated in Equations 70 to 75. Equations 72 and 75 specify the maximum dynamic sector count allowable based on conflicts and minimum separation, respectively. Since the ratio of effective sector area to total sector area will always range between 0 and 1 Equation 75 is more critical than Equation 72.

$$E(N_{conflicts}) \leq E(N_{conflicts})_{max} \tag{70}$$

$$\left(\frac{N(t)}{A_{effective}(t)}\delta^2\pi r^2\right)N(t) \leq \left(\frac{N_{capacity}}{A_{sector}}\delta^2\pi r^2\right)N_{capacity} \tag{71}$$

$$N(t) \leq \sqrt{\frac{A_{effective}(t)}{A_{sector}}}N_{capacity} \tag{72}$$

$$\mu_\delta \geq \mu_{\delta,max} \tag{73}$$

$$\frac{1}{2\sqrt{\frac{N(t)}{A_{effective}(t)}}\delta} \geq \frac{1}{2\sqrt{\frac{N_{capacity}}{A_{sector}}}\delta} \tag{74}$$

$$N(t) \leq \frac{A_{effective}(t)}{A_{sector}}N_{capacity} \tag{75}$$

During each time period the probability of exceeding capacity is calculated using the Poisson distribution (Equation 76). If this threshold is exceeded during one 15 minute time period then the inclusion of this sector in the flow constrained area may not be necessary. For this reason an optional calculation may be performed that checks the percentage of time that the probability threshold is exceeded (Equation 77).

$$P\left(N(t) > N_{effective}(t)\right) = 1 - \sum_{i=0}^{\lfloor N_{effective}(t)\rfloor}\frac{N(t)^i e^{-N(t)}}{i!} \tag{76}$$

$$\frac{\sum_{t=1}^{T}\left(P\left(N(t) > N_{effective}(t)\right)\right) > p_{max}}{T}100\% > t_{\%,max} \tag{77}$$

7.2 Collaborative Decision-Making

Using a flow constrained area that can be communicated to airlines represents the implementation of collaborative decision-making principles. As before, airlines must submit their intentions into the ETMS system to make known their flight plan so appropriate demand calculations can be carried out. The FAA can then use a method, such as the one presented above, to determine those sectors that are anticipated to be congested. A collaborative routing scheme can then be developed and communicated to the airlines and other interested airlines. However, when a flow constrained area is communicated to the airlines the airlines should have the option of avoiding the congested airspace. In other words, the airlines would

have more freedom in the routing of their own flights outside of the flow constrained area and not be required to follow the prescribed routing through the congested airspace if this routing is undesirable.

7.3 Flight Grouping

The National Severe Weather Playbook described in the literature review (Section 2.1) is the current tool used to route flights during severe weather. The playbook does not prescribe a route for each flight but instead uses the departing airport, arriving airport, or Air Route Traffic Control Center (ARTCC) the flight crosses as filters to generate a route for groups of flights.

The routing of groups of aircraft is potentially less efficient than generating detailed flight plans for each flight. However, this is balanced against the computational speed gained by reducing the number of routes to be generated. The method presented to group flights can be set so that there is one flight per group so more efficient routing schemes can be explored if necessary.

The following nomenclature is used in the clustering algorithm:

$\mathcal{D}(A,B)$ = Distance between cluster A and B (nm^2)

$D(i,j)$ = Distance function between flight i and j used in the clustering algorithm (nm^2)

$dist(lat_i, lon_i, lat_j, lon_j)$ = Great circle distance between flight i coordinates and flight j coordinates (nm)

IC = Inconsistency coefficient

$lat1_i$ = Latitude at departing airport or specified distance outside (before) flow constrained area for flight i

$lat2_i$ = Latitude at arrival airport or specified distance outside (after) flow constrained area for flight i

$lon1_i$ = Longitude at departing airport or specified distance outside (before) flow constrained area for flight i

$lon2_i$ = Longitude at arrival airport or specified distance outside (after) flow constrained area for flight i

Hierarchical data clustering is the method used to group flights (Jain and Dubes, 1988). This method begins by having each element, in this case a flight, in its own cluster and progressively merging these clusters based on a distance function. The distance function used is based on the squared distance between the flight start and end locations where the start and end locations may also be set at a specified distance outside of the flow constrained area instead of at the airport (Equation 78). The maximum distance between a pair of flights in each cluster is used to establish the distance between clusters (Equation 79). The maximum distance clustering is known as complete clustering.

$$D(i,j) = dist(lat1_i, lon1_i, lat1_j, lon1_j)^2 + dist(lat2_i, lon2_i, lat2_j, lon2_j)^2 \quad (78)$$

$$\mathcal{D}(A,B) = max\{D(i,j) : i \in A, j \in B\} \quad (79)$$

The algorithm to cluster the flights is straightforward and is as follows. Flights are segregated by cruising altitude since the potential routes are different for those following airways at less than 18,000 feet than for flights using jet routes at 18,000 feet and above. The clustering distance between each flight is then established based on Equation 78. Each flight begins in its own cluster and is successively merged with another cluster based on the maximum distance function of Equation 79. At each iteration the two clusters with the minimum distance are merged until a stopping criterion is achieved.

The stopping criterion can be based on either a user specified maximum number of clusters or until the inconsistency coefficient (Equation 80) exceeds a threshold. Based on preliminary comparisons an inconsistency coefficient value of 0.75 produces more reasonable cluster sizes and is described in more detail here to decide if clusters E and F should be merged into a new cluster. Consider clusters A and B that have already been merged to produce cluster E and clusters C and D that have been merged to get cluster F. The mean (Equation 81) and standard deviation (Equation 82) of the maximum distance between the clusters are used to calculate the inconsistency coefficient. By convention, the inconsistency coefficient is set to 0 if there is only one flight in each of the clusters that are candidates to be merged.

$$IC = \frac{\mathcal{D}(E,F) - \bar{x}_D}{\sigma_D} \tag{80}$$

$$\bar{x}_D = \frac{1}{3}\left[\mathcal{D}(A,B) + \mathcal{D}(C,D) + \mathcal{D}(E,F)\right] \tag{81}$$

$$\sigma_D = \sqrt{\frac{1}{3}\left[(\mathcal{D}(A,B) - \bar{x}_D)^2 + (\mathcal{D}(C,D) - \bar{x}_D)^2 + (\mathcal{D}(E,F) - \bar{x}_D)^2\right]} \tag{82}$$

The clustering method presented is computationally efficient and straightforward to implement in software but there are some notable drawbacks to the method. Once a flight has been added to a cluster it can only be merged into larger clusters but cannot be transferred to another cluster. For this reason the clustering method is not "optimal" in any sense. Also, the distance function (Equation 78) and the cluster cutoff criterion (Equation 80) are both arbitrary and may be inferior to other metrics.

7.4 Graph Theoretic Network Generation

Graph theory and associated methods are used to efficiently generate alternative routing schemes for each of the weather scenarios. A thorough treatment of graph theory is provided in Ahuja et al. (1993) and Bazaraa et al. (2005). The following describes the use of airspace elements such as airports, jet routes, etc. to generate a network suitable for shortest path calculations. This analysis is also intended to reduce much of the computational effort required in the calculation of sector entry and exit locations by preprocessing this information. Similar techniques to preprocess sector occupancy is presented elsewhere (e.g. Bertsimas and Patterson (1998, 2000)). The geometric calculations for sector occupancy can be extensive and any technique to reduce geometric calculations has the potential to decrease computational time.

Navigational aids (NAVAIDs) and airspace fixes used to define jet routes and airways are idealized as nodes, or vertices, in the network. The network under construction is three-dimensional so the nodes need an additional dimension of altitude to add to the NAVAIDs and fixes. Nodes are constructed at the sector floor, sector centroid, and sector ceiling to facilitate sector occupancy calculations (Figure 34). Airports are also idealized as nodes but only one node per airport is required with an altitude at ground level.

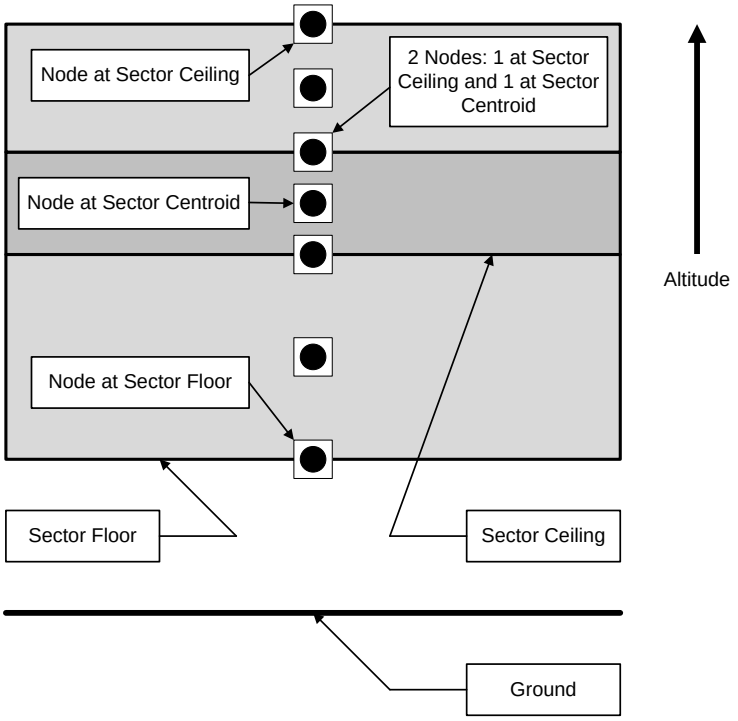

Figure 34: Altitude of NAVAIDs and Fixes Relative to Sector Floor, Centroid, and Ceiling.

As part of the preprocessing of sector entry and exit calculations nodes are added at sector boundaries along the path of jet routes or airways depending on altitude (Figure 35). Nodes are added at the sector floor, sector centroid, and sector ceiling similar to the case of nodes for NAVAIDs and fixes.

Directional links are then added to join nodes together and fully connect the network. Airport nodes are connected to the lowest NAVAID/fix nodes on airport departure paths. Vertical links of zero length are added between the nodes for the same NAVAID/fix shown in Figure 34. Horizontal links are added along the path of jet routes and low altitude airways to connect the nodes shown in Figure 35. One day of flight plans is then examined to check for additional links required for complete connectivity.

Both the nodes and links have associated properties to facilitate routing calculations (Tables 30 and 31). The travel time for each link considers changes in wind conditions at 15 minute intervals and the aircraft performance using the link. Each flight group has a link travel time tailored to the aircraft performance of the group.

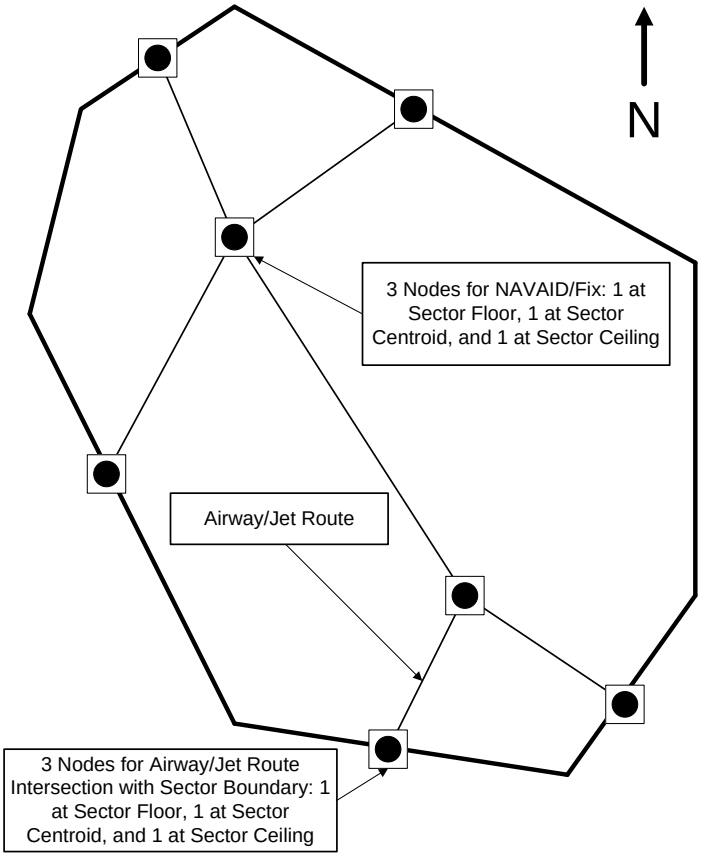

Figure 35: Preprocessing of Intersection of Airways and Jet Routes with Sector Boundaries as idealized by Nodes.

Table 30: Properties Associated with Node Elements.

Property	Description
Latitude	Latitude of the node
Longitude	Longitude of the node
Altitude	Altitude of the node
Type	Type of the node (airport, sector floor, sector centroid, sector ceiling, boundary, etc.)
Sector	Sector containing the node
Out Link List	List of links exiting the node used for shortest path calculations
Identifier	Identifier of NAVAID or airspace fix associated with this node
Cost at Node	Array of values used in shortest path algorithm calculations

Table 31: Properties Associated with Link Elements.

Property	Description
From Node	Node at tail of this link
To Node	Node at head of this link
Length	Preprocessed length of this link (nm)
Azimuth	Preprocessed azimuth of this link (degrees)
Sector	Sector containing the link
Floor Altitude	Altitude at the sector module floor. Sectors may not have a uniform sector floor if composed of several modules.
Ceiling Altitude	Altitude of the sector module ceiling. Sectors may not have a uniform sector ceiling if composed of several modules.
Identifier	Unique identifier for the link
Type	Link type (airport connector, jet route, victor airway, etc.)
Travel Time	Travel time array for each time period to account for wind conditions and aircraft performance

The network used in this dissertation, after adding all of the nodes and links described, has a total of 297,879 nodes and 638,343 links. For computational efficiency reasons only the east coast of the U.S. contains a detailed network, the rest of the U.S. and international airspace contains a simplified representation with one node per NAVAID or fix.

7.5 Incremental Assignment

The incremental assignment method is a simple procedure whereby one group of flights is assigned to the network, the sector-based demand is updated, links are set to unavailable if the capacity threshold is exceeded, then the next group of flights is assigned. The strategy and order of the selection of groups will have an impact on the efficiency of the routing scheme so this issue should be the focus of further study.

Assignment of a group of flights begins by finding the shortest path from the beginning to the end of the flight group as defined in Section 7.3. A Fibonacci heap implementation of Dijaskra's algorithm (Algorithm 1) is used in the shortest path calculations (Ahuja et al., 1993). This shortest path is checked against all of the weather scenarios. Recall that multiple weather scenarios are generated to account for weather uncertainty (Figure 17). Paths are generated until all of the weather scenarios have been satisfied.

Demand for the sector is updated using the method described in Section 5 and all links in the sector are removed from further consideration if the probability threshold of exceeding capacity is reached. Links are "removed" by setting the travel time to a large value, in this case infinity.

7.6 k-Shortest Paths and Integer Programming (APCDM)

A potentially more robust and computationally expensive method is to generate several sub-optimal routes for each flight group and use integer programming methods to select the best route. The Airspace Planning and Collaborative Decision-Making (APCDM) model described in the literature review (Section 2.9) is used to select from the alternative routes.

The alternative routes are generated using the same algorithm as for the incremental assignment. In this case links are removed from the network for each flight group to get a disjoint path. The three end links at either end are not removed to ensure that a valid path

Algorithm 1 Heap Dijkstra

```
Create heap
for all nodes in node set do
    Set node distance label to ∞
end for
Set distance label of start node to 0
Set predecessor node of start node to NULL
Insert start node into heap
while heap is not NULL do
    Find, delete, and select minimum node in heap
    for each link in outlink list of node do
        Set value to distance label of the node + length of the link
        if value < distance label of link to node then
            if distance label of link to node = ∞ then
                Insert to node into heap
            else
                Decrease Fibonacci heap by value
            end if
            Set distance label of the link to node to value
            Set link to node predecessor to node
        end if
    end for
end while
```

is available at each iteration. The classical k-shortest path methods are not considered here due to the possibility of cycles for non-disjoint versions and potentially not being able to generate enough paths using the disjoint methods.

The k-shortest path methods are intended as a comparison for the incremental methods that are faster but may not be as efficient. In the validation section it is demonstrated how efficient the incremental methods are compared to the k-shortest path and integer programming methods.

8 SIMULATION AND VALIDATION

In this section the computational tool AirPlanJ is introduced that implements the methods described in Sections 5 to 7. This tool is used to generate routing schemes for a weather-induced capacity restriction scenario. The routing scheme is then validated in an airspace simulation tool RAMS introduced in Section 3.

8.1 Airspace Planning in Java (AirPlanJ)

The Java tool that performs the airspace planning calculations is referred to as AirPlanJ. The software is currently text-based and non-interactive. AirPlanJ begins by reading all of the relevant data including: airports, navigational aids (NAVAIDs), airspace fixes, standard terminal arrivals (STARs), departure paths (DPs), BADA aircraft performance files, jet routes, airways, airspace sector geometries, preprocessed link and node definitions, kernel-smoothed departure and en route errors, wind forecasts, weather forecasts, and FAA ETMS flight information both active and planned. Baseline demand is established and weather scenarios are randomly generated.

The methods described in Section 7 are applied to generate both an incremental solution and an input file for the Airspace Planning and Collaborative Decision-Making Model (APCDM). These output files are used as an input into the simulation methods described in this section. Appendices D, E, and F include a description of the input data, source code, and manual respectively for AirPlanJ.

8.2 Weather Capacity Restriction Scenario

The date of July 27, 2005 at 2100 UTC to 2300 UTC is used to compare the baseline ETMS flight plan routing, the integer programming method of selecting from alternative trajectories (APCDM), and the incremental method proposed in this dissertation. Severe thunderstorms occurred along the east coast of the U.S. during this date and time which had a large impact on flights and cancellations (Figure 36). The 1-hour National Convective Weather Forecast (NCWF) at 2100 UTC is used in anticipation of a severe weather event from 2200-2300 UTC. The time from 0000 UTC to 2200 UTC on July 27, 2005 is used as a warm-up for the simulation and the scenario.

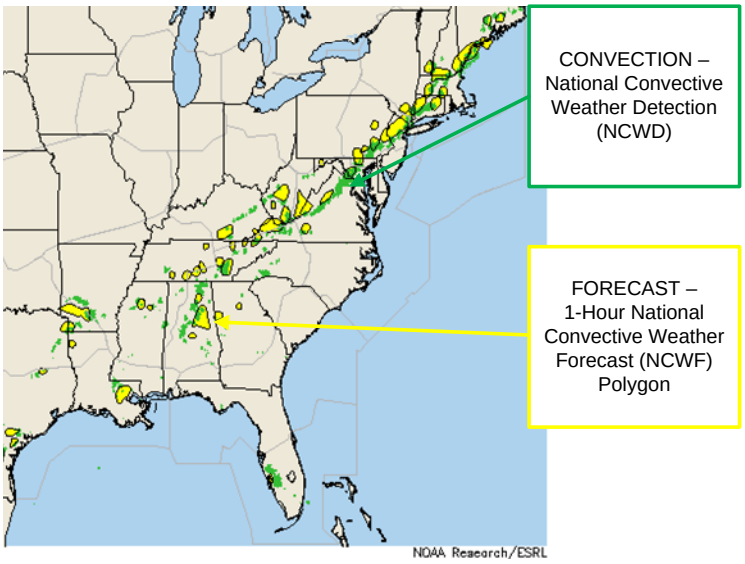

Figure 36: Severe Thunderstorms Impacting Airspace Along East Coast of U.S. on July 27, 2005 at 2200 UTC (NOAA Earth System Research Laboratory, 2007).

The magnitude of aircraft delays during this day are large. For example, at New York LaGuardia Airport (LGA) the average delay is 60 minutes for departures and 74 minutes for arrivals for flights impacted by the severe weather. Newark Airport, a nearby facility, reports similar average delays of 75 minutes for departures and 81 minutes for arrivals during the severe weather time period. These delays are obtained from the FAA's Aviation System Performance Metrics (ASPM) accessible from http://aspm.faa.gov/aspm/.

The AirPlanJ software is used to analyze the demand and capacity for the following east coast air route traffic control centers (ARTCCs) during the 2200 UTC to 2300 UTC time period: Chicago (ZAU), Indianapolis (ZID), Atlanta (ZTL), Jacksonville (ZJX), Miami (ZMA), Washington (ZDC), Cleveland (ZOB), New York (ZNY), and Boston (ZBW). A total of 49 airspace sectors are projected to experience an unacceptable level of airspace congestion as defined in Section 7.1 and are added to the flow constrained areas (FCAs).

The flights projected to enter one or more of the flow constrained areas are extracted for analysis. A total of 13,365 flights are extracted which are divided into 1,938 flights cruising at less than 18,000 ft. and the remainder of 11,427 flights projected to be cruising at or above 18,000 ft. The 18,000 ft. cutoff is chosen since flights cruising at lower altitudes may

require a different routing scheme due to the altitude limits of airways (less than 18,000 ft.) and jet routes (greater than or equal to 18,000 ft.). All of this analysis is automated in the AirPlanJ software.

The baseline routes used for comparison are the flight plans reported to the Enhanced Traffic Management System (ETMS, Appendix A). In general, if a flight is part of the National Severe Weather Playbook routing (Section 2.1) then the flight plan will be updated to reflect the selected playbook route. The reporting of playbook routes is not consistent across all FAA facilities and some flights will not have the updated route reported to ETMS. No check is made in this analysis as to whether all flights conform to the playbook since access to historical playbook routes is unavailable.

Alternative routes for the integer programming analysis are generated using the k-shortest path technique described in Section 7.6. Routing is performed for each of the 4,619 clusters of flights that are divided into 555 clusters for flights cruising below 18,000 ft and 4,064 clusters for flights cruising at or above 18,000 ft. A total of 10 routes are generated around forecasted convective weather locations for each of the clusters. Only one deterministic forecast is used to generate the routes and the capacity of sectors is not considered in the generation of the routes.

The Airspace Planning and Collaborative Decision-Making Model (APCDM) integer programming parameters used in this analysis are the same as those used in the preliminary validation (Table 15). The alternative of cancelling a flight is not considered to better compare the integer programming selection of a route to the incremental method. In the incremental method no flights are cancelled and instead flights are routed through longer paths further from their original flight plan. It is assumed in a collaborative decision-making framework that the cancellation decision should be left to the airspace user and not the FAA. Also, the conflict resolution and equity constraints are removed from consideration. The potential for conflicts is already explicitly considered in the development of sector capacity so considering it again here would be redundant. The equity constraints are an important consideration but there are several small operators and general aviation aircraft for which the equity constraints cannot be applied. Instead equity between users is checked after the simulation.

The incremental method presented in this dissertation generates routes for each of the clusters considering twenty (20) randomly generated convective weather scenarios. The scenarios are grouped by airspace capacity impact to reduce the number of routes per cluster to four (4). In the incremental method the demand for airspace sectors are updated after each of the clusters is assigned. Upon reaching capacity the airspace sector does not allow additional flights to enter and longer routes are generated for the flights in the cluster. Each of the weather scenarios have different dynamic sector capacities that are considered in the generation of routes.

On an Intel Core 2 Quad Q6600 2.4 GHz processor the k-shortest path calculations requires 47 minutes and the incremental assignment method requires 58 minutes for this scenario. To solve the integer programming problem using CPLEX on a AMD Athlon 64 X2 4400+ 2.2 GHz dual core processor requires 26 minutes.

8.3 RAMS Simulation Tool to Compare Routing Methods

The Reorganized Air traffic control Mathematical Simulator (RAMS) is used to compare the routing methods (ISA Sofware Ltd., 2004). RAMS is a fast-time analysis tool that includes the following simulated objects relevant to this analysis: air traffic controllers, airspace sectors, aircraft with associated performance characteristics, and en route restrictions.

In RAMS air traffic controllers are able to deconflict flights and have associated workload models. Air traffic controllers and their controlled airspace sectors also have defined capacities in terms of aircraft counts. When a sector exceeds capacity the workload of the controller is increased and additional reporting is included in the output files but airborne flights are not denied entry into the sector. Unfortunately, sector capacity is static and cannot be dynamically reassigned. To account for this the lowest sector capacity calculated in the time period of 2200 UTC to 2300 UTC is used in the simulation.

One of the capabilities of RAMS exploited in Section 3 is the ability to close regions of en route airspace (Figure 11). This functionality is exploited to model severe weather locations obtained from the observed convection. RAMS is able to dynamically route flights around a closed region of airspace. If an aircraft is able to climb over the convective weather system then a routing above the airspace restriction is explored.

Another important consideration in this dissertation is the modeling of uncertain aircraft departures. To this end the departure times used in the simulation are observed departures obtained from the ETMS departures file (Appendix A).

8.4 Simulation Results

The results from RAMS is used to compare the routing schemes based on: the number of times en route sectors exceed capacity, count of conflict resolution maneuvers issued by air traffic controls, count of weather avoidance maneuvers, and en route delay per flight. All of the metrics are measures of en route air traffic controller workload. En route delay, while not an intuitively obvious workload metric, requires controllers to monitor flights for additional time which increases workload. En route delay also imposes costs on airlines in the form of fuel and employee expenses. All analysis is based on the 2200 UTC to 2300 UTC hour so even though 13,365 flights are simulated only 1,779 to 2,351 flights are airborne in the constrained airspace during this hour. The time before 2200 UTC is used as a warm-up period in the simulation and statistics from this time period are not collected.

The incremental routing method produces routes for 5 randomly generated weather groups. The routes generated from each of the weather groups are similar and the summary results are indistinguishable. So the results presented in this section apply to each of the routing schemes generated by the incremental method.

There are 432 sectors modeled in the simulation for which there are 1728 sector analysis time periods. Each sector analysis time period lasts 15 minutes. Figure 37 shows that the routing methods do not improve on the flight plan routes. This is due to the uncertainty associated with the flight departure time. However, when the sector capacity is exceeded the amount by which the capacity is exceeded, in terms of the count of flights in the sector, is reduced in both the integer programming and incremental routing cases as shown in Figure 38.

The use of a ground holding algorithm, such as those discussed in Section 2.2, are used operationally to minimize the number of times that en route sectors and airports exceed capacity. These algorithms would complement the methods discussed in this dissertation by specifying the amount of time a flight should be held on the ground to satisfy capacity

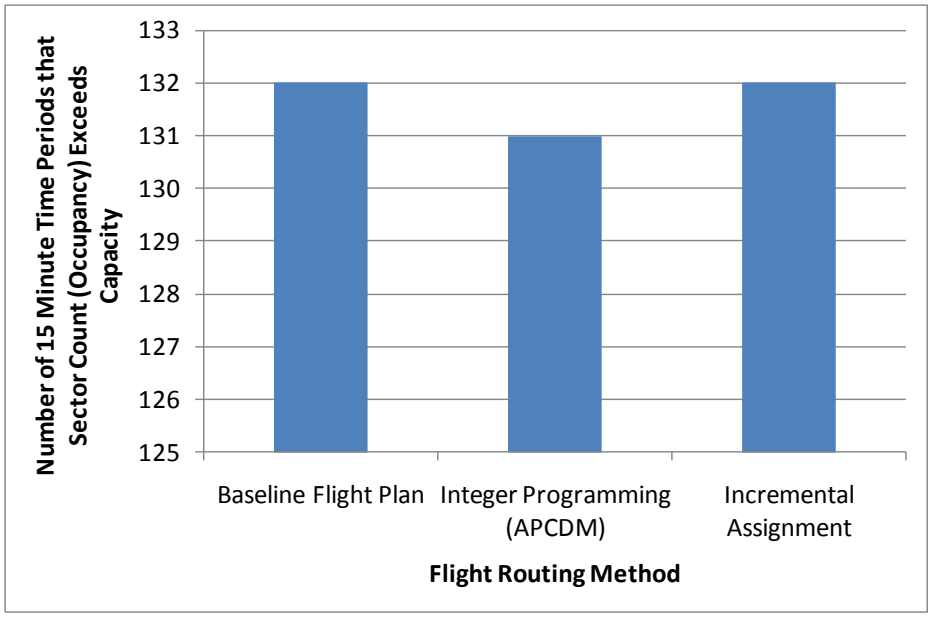

Figure 37: Count of 15 Minute Time Periods that Flights in a Sector Exceed Capacity by Routing Method.

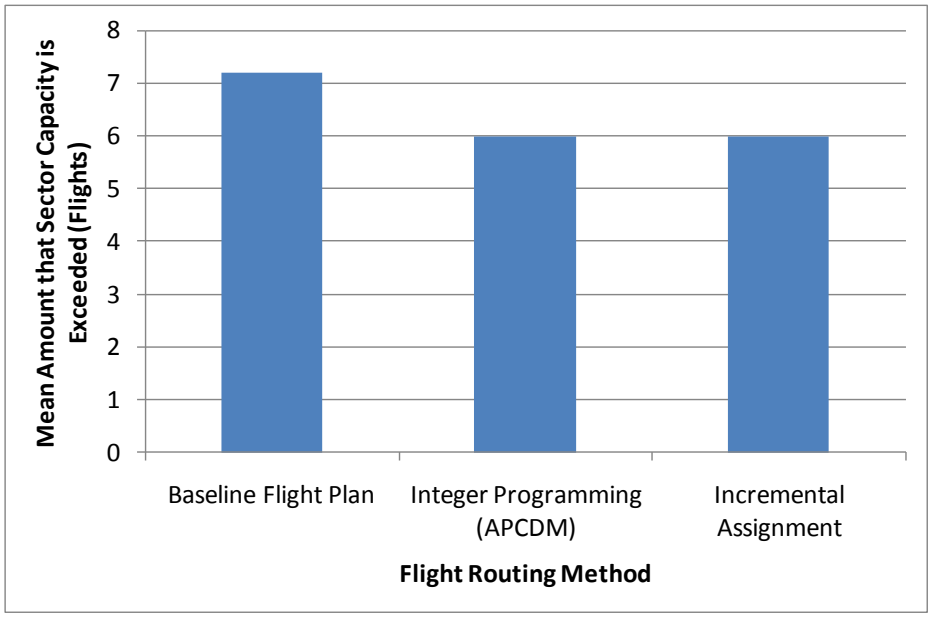

Figure 38: Mean Amount that Sector Capacity is Exceeded by Routing Method.

constraints. Ground holding decisions are typically made just prior to departure and could potentially prevent many of the capacity violations reported in the simulation.

There are two parameters in the model discussed in Section 7.1 that affect the percentage of time that the sector capacity is exceeded. The thresholds for the probability of exceeding capacity ($p_{max} = 0.55$) and the percentage of time that a sector exceeds the probability threshold ($t_{\%,max} = 0.55$) may be reduced to prevent capacity imbalances. When these parameters are reduced the flights will be routed further from the flow constrained area resulting in increased flight delays. These tradeoffs are considerations in the selection of appropriate model parameters.

The RAMS simulation software is able to detect potential loss of separation, known as conflicts, and resolve the conflicts using a rule-base set. Figure 39 shows that the number of required conflict resolution maneuvers issued by air traffic controllers are reduced by 8.5% in the integer programming case and by 11.9% in the incremental case.

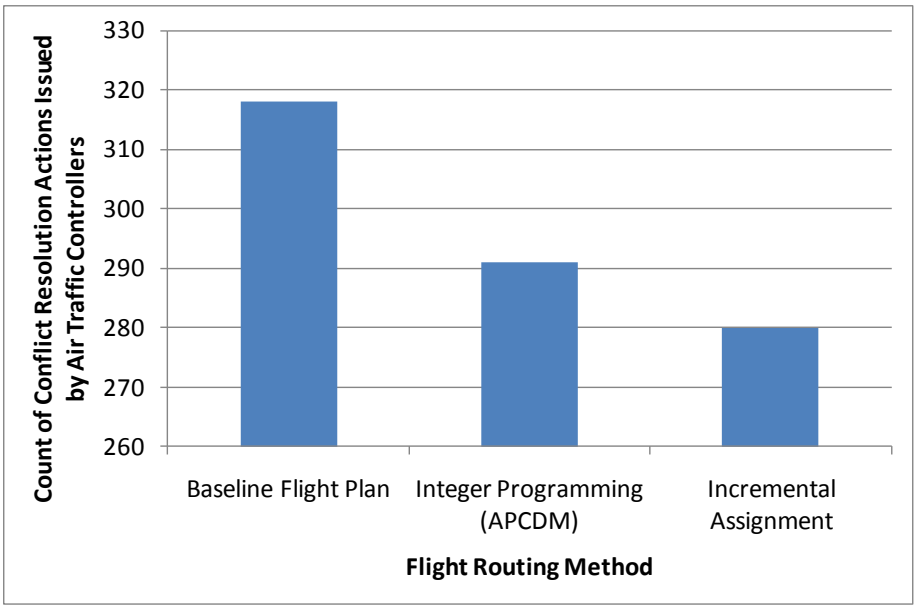

Figure 39: Count of Conflict Resolution Maneuvers Issued by Air Traffic Controllers by Routing Method.

RAMS air traffic controllers are also able to route flights around restricted airspace such

as convective weather. If a flight is projected to cross the boundary of the weather system restriction RAMS searches for the shortest route to avoid the weather and issues appropriate avoidance maneuvers. Figure 40 shows that both routing methods reduce the number of flights that are impacted by the severe convective weather by 10%.

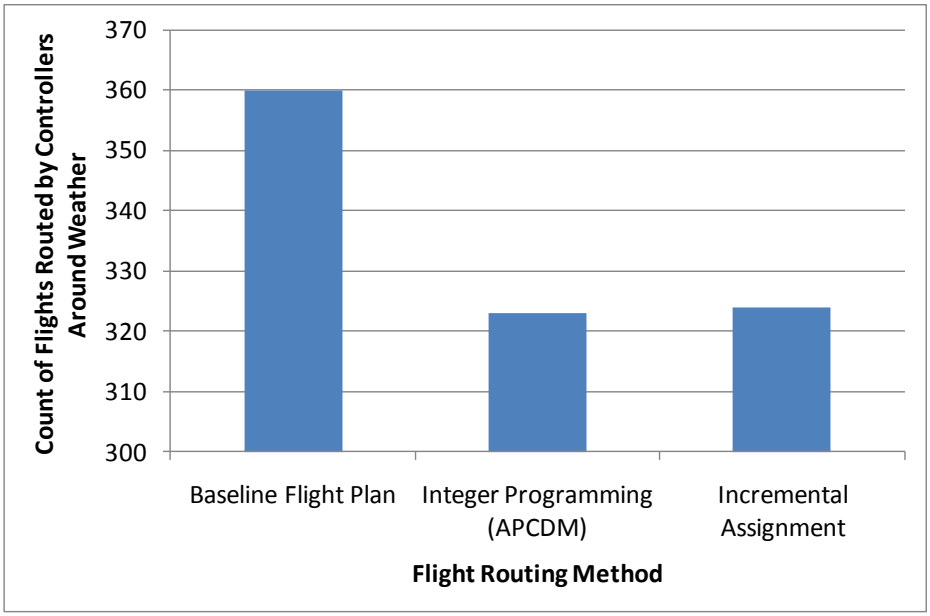

Figure 40: Count of Weather Avoidance Maneuvers Issued by Air Traffic Controllers by Routing Method.

The flight plans without weather and conflicts is used as a basis for establishing en route delays. En route delay is that portion of the excess flight time a flight receives between the origin airport terminal area and destination airport terminal area, exclusive. The negative delays shown in Figure 41 indicate that the two routing methods results in flight times that are less than the undelayed flight plans. The integer programming method improves on the flight plan by 6.2 minutes per flight and the incremental routing method improves on the flight plan by 6.0 minutes per flight.

8.5 Equity Considerations

In this section a comparison is presented between airborne travel time for the flight plan routes, integer programming selected routes, and the routes generated by incremental as-

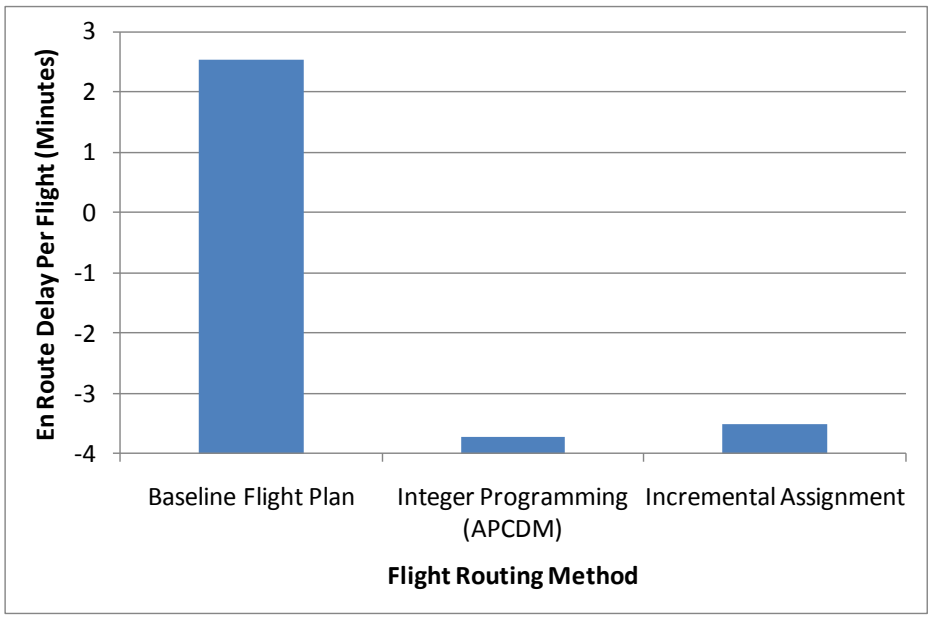

Figure 41: En Route Delay per Flight by Routing Method.

signment. This comparison checks that no airlines are unfairly punished by participating in a routing scheme. The results shown in Table 32 indicate that only two airlines, JIA (PSA Airways/US Airways Express) and FDX (FedEx), increase the average travel time per flight over the flight plan. There is an issue that some airlines, such as American (AAL) and United (UAL), improve their travel times by a greater value than other airlines and may gain a competitive advantage in the long run. However, this may be due to the airport market pairs that these airlines are serving at the time of the convective weather event and the proximity of the airports to the convective weather system.

Table 32: En Route Delay by Airline.

Airline Rank	Airline	Count of Airborne Airline Flights	Routing Method Delay (minutes per flight)		
			Flight Plan Delay	Integer Programming Delay	Incremental Delay
1	DAL	227	2.43	-4.72	-5.15
2	AAL	152	1.63	-10.71	-10.41
3	SWA	139	-0.75	-0.54	-0.52
4	USA	137	3.00	-3.49	-1.78
5	COA	114	3.89	-7.75	-8.20
6	UAL	87	0.57	-21.5	-21.17
7	COM	79	7.26	-3.64	-1.00
8	BTA	78	2.51	-2.75	-2.35
9	NWA	70	-1.64	-2.21	-2.29
10	TRS	66	2.38	-0.41	0.29
11	CHQ	61	2.46	1.57	1.93
12	EGF	55	0.81	-4.40	-4.06
13	CAA	53	2.76	-0.55	0.15
14	JBU	38	0.8	-2.20	-1.47
15	EJA	34	0.29	-5.72	-7.07
16	ASH	31	1.47	-2.81	-1.65
17	AWE	29	1.00	0.31	-17.72
18	LOF	27	1.57	-0.76	-0.76
19	PDT	27	4.68	-2.04	-1.04
20	FLG	24	1.02	-1.45	-1.11
21	JIA	24	0.71	3.96	3.84
22	FDX	23	4.40	7.94	6.94
23	ACA	21	2.44	-4.14	-3.32
24	IDE	21	2.72	-1.40	0.07
25	OPT	20	6.11	2.53	3.38

9 CONCLUSIONS

This dissertation studies the problem of airspace flight planning during severe convective weather. The goal is to create safe and efficient routes for flights in a collaborative decision-making framework of equitable decisions for all airspace users. The proposed approach analyzes congestion at a more macroscopic level than is currently available in the literature in an effort to reduce the solution computation time. Also, additional stochastic elements are modeled to more appropriately consider randomness in the en route airspace.

A preliminary validation of an existing airspace planning model based on integer programming methods is performed to guide the development of the models presented in this dissertation. Results indicate that planning for the en route airspace during severe weather can be improved by including the following considerations. There should be a more rigorous method to define the area of impact of the severe weather systems and the flights that potentially could be impacted. Also, due to the highly uncertain nature of flight departure time and route a more detailed stochastic airspace demand model should be implemented. Sector capacity should also be random considering the current quality of convective weather forecasts at a planning horizon of 1 to 2 hours before the anticipated event.

A demand model is developed that considers departure time prediction errors and en route traversal time uncertainty. The departing airport size is the major factor in the quality of departure time prediction. In the en route airspace planned sector traversal time is the most reliable predictor of en route traversal time uncertainty. A method is developed that combines the departure and en route sources of error. This method is shown to reduce the prediction error for en route sector demand by 20%.

Random separation and conflict potential models are developed using principles of Poisson point processes. These models, when combined with a reduction in airspace from severe convective weather, are used to dynamically estimate sector capacity. The random separation prediction model performs well for large geographic areas but the prediction quality of the model degrades for smaller geographic areas. A sector-based correction factor is shown to be consistent across multiple days indicating that the random separation model can be a good approximation. The conflict potential model is validated indirectly since conflict

resolution maneuvers are not available from historical data.

A graph theoretic framework that integrates the random demand and capacity models is used to develop routing schemes. A computationally efficient incremental assignment method is compared to an integer programming method that selects flights from a list of candidates. The comparison is based on the fast-time airspace simulator RAMS. Results of the simulation indicate that both the integer programming method and the incremental routing method can reduce the average en route travel time while reducing the number of conflict resolutions and weather avoidance maneuvers issued by en route air traffic controllers.

The number of 15 minute time periods that en route airspace sectors exceed capacity is not improved by the integer programming or incremental routing methods due to the uncertainty in aircraft departure time. However, the amount by which sector capacity is exceeded is reduced by both proposed routing methods over the original flight plan routes. The use of a ground holding algorithm could be used to complement the airspace planning model and prevent demand to capacity imbalances in the en route airspace and at arrival airports.

The RAMS simulation software is also used to track the count of conflict resolution maneuvers issued by air traffic controllers to prevent violation of minimum separation standards. By routing flights away from highly congested airspace the integer programming method is able to reduce the count of conflict resolution actions by 8.5% over the original flight plans. The incremental solution is found to improve on the flight plans by 11.9%, which is larger improvement compared to the integer solution in this case. The count of weather avoidance maneuvers is also tracked using the RAMS simulation software similar to conflict resolutions. The integer programming and incremental methods both result in a 10% reduction in weather avoidance maneuvers over the baseline flight plan case.

The use of the flight plan trajectories without conflict resolution or weather avoidance maneuvers is used to establish a baseline for en route delay. En route delay is that component of excess flight time that an an aircraft experiences between the top of climb at the origin airport and the top of descent at the destination airport. The average en route flight time is reduced by 6 minutes for both the integer programming and incremental methods.

Most airlines improve their airborne time over the originally submitted flight plan. How-

ever, an equity issue is that some airlines reduce their travel time more than other airlines. Based on the simulation results there is no significant difference in the quality of the results between the incremental routing method and the integer programming method of selecting a flight plan from a set of alternatives.

10 RECOMMENDATIONS

The following future work is recommended to improve en route airspace planning models.

- Development of a model to predict altitude. Aircraft may change cruising altitude several times during the course of a flight which can have an impact on demand for en route airspace sectors. In this dissertation it is assumed that flights use a constant cruising altitude which introduces an error component into the en route airspace demand calculations. A more robust altitude model that predicts aircraft climbing or descending to avoid congested airspace, turbulence, or conflicting aircraft would reduce the unexplained error in the demand calculations and improve decisions made by an airspace planning model.

- Methods with the ability to identify flights that will deviate from their planned route would be useful. The inability to predict the complete route for a flight prevents accurate sector demand calculations. Air traffic managers and air traffic controllers routinely alter the planned route of a flight to avoid congested airspace, turbulence, convective weather, or pilot requests for a more direct route which introduces an unexplained error term to the en route airspace demand model. This is the lateral equivalent to the suggested altitude prediction model and has the similar potential to reduce the unexplained error in demand calculations and improve airspace planning decisions.

- An investigation into other spatial point processes that can model interactions between points (flights). These models could potentially be used to provide better estimates of aircraft separation and conflict potential in the en route airspace. These more advanced methods cannot be solved analytically but may eliminate the need for correction factors. The point process models have the potential to improve en route sector capacity estimates by considering such factors as separation and conflict potential in a human factors model, an improvement over the current practice of using peak count of aircraft only.

- Comparison of additional heuristics to generate routing schemes. A different order of cluster selection or re-optimization methods could be more efficient than the incremen-

tal demand assignment scheme presented in this dissertation. The integer programming method is limited in that it cannot consider congestion in the generation of alternative routes. The incremental method also does not produce an optimal solution. Other techniques to generate routes have the potential to further reduce en route travel time.

- Simulation of additional en route weather scenarios. Simulation is time consuming but necessary to gain a better understanding of the en route airspace during severe convective weather. Simulation is also vital in demonstrating that a proposed model will improve operations without introducing undesired effects that were not considered during model development. If a model is to be used operationally then a range of conditions must be considered in the simulation to check for those conditions where the airspace planning model may actually degrade operations.

11 REFERENCES

Ahuja, R., Magnanti, T., and Orlin, J. (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, N.J.

Arad, B. (1964). The control load and sector design. *Journal of Air Traffic Control*, pages 12–31.

Ball, M., Hoffman, R., Hall, W., and Muharremoglu, A. (1998). Collaborative decision making in air traffic management: A preliminary assessment. Technical Report RR-99-3, NEXTOR, University of Maryland, College Park, MD.

Ball, M., Hoffman, R., Knorr, D., Wetherly, J., and Wambsganss, W. (2003a). A general approach to equity in traffic flow management and its application to mitigating exemption bias in ground delay programs. In *Proc. 5th USA/Europe Air Traffic Management R'&'D Seminar*, Budapest, Hungary. EUROCONTROL/FAA. http://atm2003.eurocontrol.fr.

Ball, M., Hoffman, R., Odoni, A., and Rifkin, R. (2003b). A stochastic integer program with dual network structure and its application to the ground-holding problem. *Operations Research*, 51(1):167–171.

Barnier, N. and Brisset, P. (2004). Graph coloring for air traffic flow management. *Annals of Operations Research*, 130(1-4).

Bayen, A., Grieder, P., Meyer, G., and Tomlin, C. (2005). Lagrangian delay predictive model for sector-based air traffic flow. *Journal of Guidance, Control, and Dynamics*, 28(5):1015–1026.

Bazaraa, M., Jarvis, J., and Sherali, H. (2005). *Linear Programming and Network Flows*. Wiley-Interscience, Hoboken, N.J., 3rd edition.

Beatty, R., Hsu, R., Berry, L., and Rome, J. (1999). Preliminary evaluation of flight delay propagation through an airline schedule. *Air Traffic Quarterly*, 7:259–270.

Bertsimas, D. and Patterson, S. (1998). The air traffic flow management problem with enroute capacities. *Operations Research*, 46(3):406–422.

Bertsimas, D. and Patterson, S. (2000). The traffic flow management rerouting problem in air traffic control: A dynamic network flow approach. *Transportation Science*, 34(3):239–255.

Canadian Aviation Safety Board (1990). Report on a special investigation into air traffic control services in canada. Technical Report Report No. 90-SO001, Supply and Services Canada. Catalog No. TU4-5/1990E.

Carlson, P. (2000). Exploiting the opportunities of collaborative decision making: A model and efficient solution algorithm for airline use. *Transportation Science*, 34(4):381–393.

Champeney, D. (1987). *A Handbook of Fourier Theorems*. Cambridge University Press, New York, N.Y.

Chang, K., Howard, K., Oiesen, R., Shisler, L., Tanino, M., and Wambsganss, M. (2001). Enhancements to the FAA ground-delay program under collaborative decision making. *Interfaces*, 31(1):57–76.

Chatterji, G. and Sridhar, B. (2001). Measures for air traffic controller workload prediction. In *Proceedings of the 1st AIAA Aircraft Technology, Integration, and Operations Forum*, Los Angeles, CA.

Collaborative Forum of Air Transport Stakeholders (2003). Fast Facts. Available from Forum Members (e.g. www.iata.org).

Davis, T., Isaacson, D., and Robinson, J. (1997). Operational field test results of the passive final approach spacing tool. In *Proceedings of the IFAC 8th Symposium on Transportation Systems*, Chania, Greece.

DOT (1997). Memorandum on departmental guidance for valuation of travel time in economic analysis. Technical report, U.S. Department of Transportation.

Dugail, D., Feron, E., and Bilimoria, K. (2002). Conflict-free conformance to en route flow rate constraints. In *AIAA Guidance, Navigation, and Control Conference*, Monterey, California.

Endsley, M. and Rodgers, M. (1996). Attention distribution and situation awareness in air traffic control. In *Proceedings of the 40th Annual Meeting of the Human Factors Society*.

Erzberger, H. (1995). Design principles and algorithms for automated air traffic management. AGARD Lecture Series 200, Brussels, Belgium, http://www.ctas.arc.nasa.gov/.

Erzberger, H., McNally, B., Foster, M., Chiu, D., and Stassart, P. (1999). Direct-To tool for en-route controllers. In *ATM 99: IEEE Workshop on Advanced Technologies and their Impact on Air Traffic Management in the 21st Century*, Capri, Italy.

Erzberger, H., Paielli, R., Isaacson, D., and Eshow, M. (1997). Conflict detection and resolution in the presence of prediction error. In *1st USA/Europe Air Traffic Management R&D Seminar*, Saclay, France.

Eurocontrol (2003). User manual for the base of aircraft data (BADA) revision 3.5. Technical report, European Organisation for the Safety of Air Navigation.

Evans, J., Allan, S., and Robinson, M. (2004). Quantifying delay reduction benefits for aviation convective weather decision support systems. In *Eleventh Conference on Aviation, Range and Aerospace Meteorology*, Hyannis, MA.

FAA (1983). Advisory circular 00-24b thunderstorms. Technical Report 00-24B, Federal Aviation Administration.

FAA (2004a). Administrator's Fact Book. http://www.atctraining.faa.gov/factbook/. Available quarterly.

FAA (2004b). National airspace system resource (NASR) subscriber CD. Technical report, Aeronautical Information Services Division, Federal Aviation Administration. February 19, 2004.

FAA (2005). FAA Operations and Performance Data, Aviation Systems Performance Metrics (ASPM). http://www.apo.data.faa.gov/aspm/entryASPM.asp. Accessed October 3, 2005.

FAA (2006a). FAA aerospace forecasts: Fiscal years 2006-2017. In *31st Annual FAA Aviation Forecast Conference*.

FAA (2006b). FAA order 7110.65 air traffic control. Technical report, Federal Aviation Administration. www.faa.gov/atpubs/ATC/.

FAA (2006c). FAA order 7210.3 facility operation and administration. Technical report, Federal Aviation Administration. www.faa.gov/atpubs/FAC/.

FAA (2006d). Interface control document for substitutions during ground delay programs, ground stops, and airspace flow programs. Technical report, Federal Aviation Administration Air Traffic Control System Command Center. Version 3.0.

FAA (2006e). National severe weather playbook. http://www.fly.faa.gov/PLAYBOOK/pbindex.html. Accessed July 9, 2006.

FAA (2007). FAA aerospace forecasts: Fiscal years 2007-2020. In *32nd Annual FAA Aviation Forecast Conference*.

FAA and ARP Consulting, L.L.C. (2000). 2000 aviation capacity enhancement plan. Technical report, Federal Aviation Administration.

Flener, P., Pearson, J., Agren, M., Garcia-Avello, C., Celiktin, M., and Dissing, S. (2007). Air-traffic complexity resolution in multi-sector planning. *Journal of Air Transport Management*, 13(6):323–328.

Forman, B., Wolfson, M., Hallowell, R., and Moore, M. (1999). Aviation user needs for convective weather forecasts. In *American Meteorological Society 79th Annual Conference*, Dallas, TX.

Free Flight Phase I Office (1999). An operational assessment of collaborative decision making in air traffic management. Technical Report R90145-01, Federal Aviation Administration.

Green, S. and Vivona, R. (2001). En-route descent advisor concept for arrival metering. In *AIAA 2001-4114, Guidance, Navigation, and Control Conference*, Montreal, Canada.

Greene, W. (2003). *Econometric Analysis*. Prentice Hall, Upper Saddle River, N.J., 5 edition.

Gwiggner, C. and Duong, V. (2006). Average, uncertainties and interpretation in flow planning. In *2nd International Conference on Research in Air Transportation (ICRAT)*, Belgrade, Serbia.

ICAO (1998). Economic contribution of civil aviation. Available from International Civil Aviation Authority at www.icao.int.

ISA Sofware Ltd. (2004). RAMS plus user manual release 5.20. Technical report.

Isaacson, D. and Robinson, J. (2001). A knowledge-based conflict resolution algorithm for terminal area air traffic control advisory generation. In *AIAA 2001-4114, Guidance, Navigation, and Control Conference*, Montreal, Canada.

Jain, A. and Dubes, R. (1988). *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, N.J.

Kay, M., Mahoney, J., and Hart, J. (2006). An analysis of CCFP forecast performance for the 2005 convective season. In *12th Conference on Aviation, Range, and Aerospace Meteorology*, Atlanta, GA. American Meteorological Society.

Krozel, J., Rosman, D., and Grabbe, S. (2002). Analysis of en-route sector demand error sources. In *AIAA Guidance, Navigation, and Control Conference*, Monterey, California.

Kuchar, J. and Yang, L. (2000). A review of conflict detection and resolution modeling methods. *IEEE Transactions on Intelligent Transportation Systems*, 1(4):179–189.

Landry, S., Farley, T., Foster, J., Green, S., Hoang, T., and Wong, G. (2003). Distributed scheduling architecture for multi-center time-based metering. In *AIAA Aircraft, Technology, Integration and Operations (ATIO) Conference*, Denver, CO.

Law, A. (2007). *Simulation Modeling and Analysis*. McGraw-Hill, Boston, MA, 4th edition.

Lee, K., Quinn, C., Hoang, T., and Sanford, B. (2000). Human factors report: TMA operational evaluations 1996 & 1998. Technical Report TM-2000-209587, NASA. http://www.ctas.arc.nasa.gov/.

Leiden, K. and Green, S. (2000). Inter-sector planning: En route controllers roles, responsibilities, and procedures. Technical Report RTO-34B, NASA Ames Research Center.

Lindsay, K., Greenbaum, D., and Wanke, C. (2005). Predeparture uncertainty and prediction performance in collaborative routing coordination tools. *Journal of Guidance, Control, and Dynamics*, 28(6):1178–1186.

Lulli, G. and Odoni, A. (2007). The european air traffic flow management problem. *Transportation Science*, 41(4):431–443.

Mahapatra, P. (1999). *Aviation Weather Surveillance Systems Advanced Radar and Surface Sensors for Flight Safety and Air Traffic Management*, volume 183 of *Progress in Astronautics and Aeronautics*. The American Institute of Aeronautics and Astronautics and The Institution of Electrical Engineers.

Mahoney, J., Henderson, J., Brown, B., Hart, J., Loughe, A., Fischer, C., and Sigren, B. (2002). The real-time verification system (RTVS) and its application to aviation weather. In *Tenth Conference on Aviation, Range, and Aerospace Meteorology*, Portland, OR. American Meteorological Society.

Majumdar, A., Ochieng, W., Bentham, J., and Richards, M. (2005). En-route sector capacity estimation methodologies: An international survey. *Journal of Air Transport Management*, 22:375–387.

Megenhardt, D., Mueller, C., Trier, S., Ahijevych, D., and Rehak, N. (2004). NCWF-2 probabilistic forecasts. In *11th Conference on Aviation, Range, and Aerospace Meteorology*, Hyannis, M.A. American Meteorological Society.

Menon, P., Sweriduk, G., and Bilimoria, K. (2004). New approach for modeling, analysis, and control of air traffic flow. *Journal of Guidance, Control, and Dynamics*, 27(5):737–744.

Menon, P., Sweriduk, G., and Bilimoria, K. (2006). Computer-aided eulerian air traffic flow modeling and predictive control. *Journal of Guidance, Control, and Dynamics*, 29(1):12–19.

Meyn, L. (2002). Probabilistic methods for air traffic demand forecasting. In *AIAA Guidance, Navigation, and Control Conference*, Monterey, California.

Mogford, R., Morrow, J., and Kopardekar, S. (1995). The complexity construct in air traffic control: A review and synthesis of the literature. Technical Report CT-TN92/22, DOT/FAA.

Mohleji, S. and Tene, N. (2006). Minimizing departure prediction uncertainties for efficient rnp aircraft operations at major airports. In *6th AIAA Aviation Technology, Integration, and Operations Conference, Collection of Technical Papers, v.2*, Wichita, Kansas.

Moller, J. and Waagepetersen, R. (2004). *Statistical Inference and Simulation for Spatial Point Processes*. Monographs on Statistics and Applied Probability 100. Chapman and Hall/CRC, Boca Raton, FL.

Mueller, K., Sorensen, J., and Couluris, G. (2002). Strategic aircraft trajectory prediction uncertainty and statistical sector traffic load modeling. In *AIAA Guidance, Navigation, and Control Conference*, Monterey, California.

National Climatic Data Center (2005). HDSS Access System: NEXRAD Level III Radar. http://hurricane.ncdc.noaa.gov/pls/plhas/has.dsselect. Accessed October, 2005.

NOAA Earth System Research Laboratory (2007). Real Time Verification System Convective Display. http://rtvs.noaa.gov/conv/op/displays/.

NOAA Research (2006). NEXRAD National Mosaic Reflectivity Images. http://www4.ncdc.noaa.gov/cgi-win/wwcgi.dll?WWNEXRAD Images2.

OAG Worldwide Limited (2004). OAG MAX User's guide edition 4.0. Technical report.

Office of Aviation Enforcement and Proceedings (2008). Air Travel Consumer Report. Available monthly at http://airconsumer.ost.dot.gov/.

Okabe, A., Boots, B., Sugihara, K., and Chiu, S. (2000). *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley Series in Probability and Statistics. John Wiley and Sons Ltd, New York, NY, 2nd edition.

Rhoda, D., Kocab, E., and Pawlak, M. (2002). Aircraft encounters with thunderstorms in en route vs. terminal airspace above memphis, tennessee. In *Tenth Conference on Aviation, Range, and Aerospace Meteorology*, Portland, OR. American Meteorological Society.

Rodgers, M. (1993). An examination of the operational error database for en-route traffic control centers. Technical report, Federal Aviation Administration, Office of Aviation Medicine.

Rodgers, M. and Drechsler, G. (1993). Conversion of the CTA, incorporated, en route operations concepts database into a formal sentence outline job task taxonomy. Technical Report DOT/FAA/AM-93/1, Civil Aeromedical Institute, FAA.

RTCA (1995). Final report of RTCA Task Force 3 - Free Flight implementation. Technical report, Radio Technical Commission for Aeronautics, Inc.

SAS Institute Inc. (2003). SAS help and documentation. Technical Report SAS 9.1.3, Cary, North Carolina, USA.

Sherali, H., Smith, J., and Trani, A. (2002). An airspace planning model for selecting flight plans under workload, safety, and equity considerations. *Transportation Science*, 36(4):378–397.

Sherali, H., Smith, J., Trani, A., and Sale, S. (2000). National airspace sectory occupancy and conflict analysis models for evaluating scenarios under the free-flight paradigm. *Transportation Science*, 34(4):321–336.

Sherali, H., Staats, R., and Trani, A. (2003). An airspace planning and collaborative decision-making model: Part I - Probabilistic conflicts, workload, and equity considerations. *Transportation Science*, 37(4):434–456.

Sherali, H., Staats, R., and Trani, A. (2006). An airspace planning and collaborative decision-making model: Part II - Data considerations, and computations. *Transportation Science*, 40(2):147–164.

Sridhar, B., Soni, T., Sheth, K., and Chatterji, G. (2006). Aggregate flow model for air-traffic management. *Journal of Guidance, Control, and Dynamics*, 29(4):992–997.

Stoyan, D., Kendall, W., and Mecke, J. (1987). *Stochastic Geometry and its Applications*. Wiley Series in Probability and Mathematical Statistics. John Wiley and Sons, Berlin, GDR.

Tsay, R. (2005). *Analysis of Financial Time Series*. Wiley-Interscience, Hoboken, NJ, 2nd edition.

Volpe National Transportation Systems Center (2002). Enhanced traffic management system (ETMS) functional description version 7. Technical Report VNTSC-DTS56-TMS-002, U.S. Department of Transportation.

Wambsganss, M. (1997). Collaborative decision making through dynamic information transfer. *Air Traffic Control Quarterly*, 4(2):109–125.

Wanke, C. and Greenbaum, D. (2007). Incremental, probabilistic decision making for en route traffic management. In *Proc. 7th USA/Europe Air Traffic Management R'&'D Seminar*, Barcelona, Spain. EUROCONTROL/FAA. atm2003.eurocontrol.fr/atm-seminar-2007.

Wanke, C., Greenbaum, D., Mulgund, S., and Song, L. (2004). Modeling traffic prediction uncertainty for traffic management decision support. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Providence, Rhode Island.

Wanke, C., Song, L., Zobell, S., Greenbaum, D., and Mulgund, S. (2005). Probabilistic congestion management. In *6th USA/Europe ATM 2005 Seminar*, Baltimore, MD.

Weber, M., Evans, J., Wolfson, M., DeLaura, R., Moser, B., Martin, B., and Bertsimas, D. (2005). Improving air traffic management during thunderstorms. In *AIAA/IEEE 24th Digital Avionics System Conference*, Washington, D.C.

Wickens, C. (1992). *Engineering Psychology and Human Performance*. Harper Collins, New York, 2nd edition.

Wickens, C., Mavor, A., and McGee, J., editors (1997). *Flight to the Future: Human Factors in Air Traffic Control*. National Academy Press.

Wiener, E. (1988). Cockpit automation. In Wiener, E. and Nagel, D., editors, *Human Factors in Aviation*, pages 433–461. Academic Press, New York.

Wong, G. (2000). The dynamic planner: The sequencer, scheduler, and runway allocator for air traffic control automation. Technical Report TM-2000-209586, NASA. http://www.ctas.arc.nasa.gov/.

APPENDIX A: ENHANCED TRAFFIC MANAGEMENT SYSTEM (ETMS) DATA DESCRIPTION

This appendix contains a description of the FAA's Enhanced Traffic Management System (ETMS) data source. ETMS data is available in a variety of formats and the one presented here is acquired from the company Flight Explorer. The data field definitions are provided by Flight Explorer and are quoted or paraphrased. Also, only the tables used in the analysis are included.

Flight Amendment (AF) File

The following fields are included in the flight amendment file:

Data Field	Description
FID	Unique identifier
Rule No	Internal (Flight Explorer) rule no.
Message Type	AF
Message Time	GMT message was sent
Center	FAA Air Route Traffic Control Center (ARTCC) generating the message
Aircraft ID	Aircraft identifier
CID	FAA computer identifier
Origin	Departure airport
Destination	Arrival airport
Heavy Flag	Indicator for a heavy jet aircraft
Total Aircraft	Aircraft count for this group
Aircraft Type	Authorized aircraft type
Aircraft Speed	Amended airspeed (knots)
Coord Location	Location where flight will enter the NAS
Coord Type	Type of coordination
Coord Time	Time flight is (or will be) transferred to the NAS
Altitude	Files cruising altitude or flight level in hundreds of feet
Route	Planned route of the flight
NRP	Not used

A sample of flight amendment data is shown below:

```
160800772,302,AF,07/27/2005,000000,K,SWA499,000,BWI,SLC,,1,,,3752N/09808W,E
   ,,,BWI./.HYS127080..JNC.J12.HELPR.SPANE4.SLC,0
160823059,302,AF,07/27/2005,000004,I,N9FE,000,CAK,MEM,,1,,,4003N/08219W,E,,
   ,CAK./.APE121014..PXV.WLDER4.MEM/0110,0
160814607,302,AF,07/27/2005,000008,G,AMT4670,000,MSP,MDW,,1,,,,,,210,,0
```

Flight Plan (FZ) File

The flight plan data fields are the same as the flight amendment data fields with the exception of the message type which is set to FZ in this case.
A sample of flight plan data is shown below:

```
160824683,0,FZ,07/27/2005,000003,S,QXE223,037,,1,DH8B,0275,CYVR,P,07/27/200
   5,005000,07/27/2005,140000,170,CYVR..SEA..HELNS.HELNS4.KPDX/0055,0
160824684,0,FZ,07/27/2005,000003,S,AMF1961,302,,1,BE99,0200,KBLI,P,07/27/20
   05,005000,07/27/2005,053000,100,KBLI.KIENO1.CVV..JAWBN.JAWBN9.KBFI,0
160824685,0,FZ,07/27/2005,000005,S,AMF1969,895,,1,BE99,0200,KBLI,P,07/27/20
   05,005000,07/27/2005,054200,100,KBLI.KIENO1.CVV..JAWBN.JAWBN9.KBFI,0
```

Flight Departure (DZ) File

The following fields are included in the flight departure file:

Data Field	Description
FID	Unique identifier
Rule No	Internal (Flight Explorer) rule no.
Message Type	DZ
Message Time	GMT message was sent
Center	FAA Air Route Traffic Control Center (ARTCC) generating the message
Aircraft ID	Aircraft identifier
CID	FAA computer identifier
Heavy Flag	Indicator for a heavy jet aircraft
Total Aircraft	Aircraft count for this group
Aircraft Type	Authorized aircraft type
Origin	Departure airport
Coord Type	Type of coordination
Departure time	Flight departure time
Destination	Destination airport
Est Arrival Time	Estimated time of flight arrival

A sample of flight departure data is shown below:

```
160813131,201,DZ,07/27/2005,000004,S,SCX288,186,T,1,B738,SEA,D,07/27/2005,0
   00000,MSP,07/27/2005,030200
160807445,201,DZ,07/27/2005,000005,L,AAL1484,673,T,1,MD83,SAN,D,07/27/2005,
   000000,ORD,07/27/2005,031800
160814366,201,DZ,07/27/2005,000007,Q,JAL73,402,H,1,B742,PHNL,D,07/27/2005,0
   00000,RJAA,07/27/2005,024100
```

Flight Cancellation (RZ) File

The following fields are included in the flight cancellation file:

Data Field	Description
FID	Unique identifier
Rule No	Internal (Flight Explorer) rule no.
Message Type	RZ
Message Time	GMT message was sent
Center	FAA Air Route Traffic Control Center (ARTCC) generating the message
Aircraft ID	Aircraft identifier
CID	FAA computer identifier
Origin	Departure airport
Destination	Arrival airport

A sample of flight cancellation data is shown below:

```
160789386,104,RZ,07/27/2005,000009,N,NPX100,780,PNE,JST
160824696,0,RZ,07/27/2005,000010,M,N193SB,236,MDQ,GSO
160774541,409,RZ,07/27/2005,000010,M,N57GA,245,FSM,PDK
```

Flight Track (TZ) File

The following fields are included in the flight track file:

Data Field	Description
FID	Unique identifier
Rule No	Internal (Flight Explorer) rule no.
Message Type	RZ
Message Time	GMT message was sent
Center	FAA Air Route Traffic Control Center (ARTCC) generating the message
Aircraft ID	Aircraft identifier
CID	FAA computer identifier
Ground Speed	Current speed of aircraft relative to the surface of the earth
Altitude	Altitude or flight level in hundreds of feet
Latitude	Latitude of aircraft at message time
Longitude	Longitude of aircraft at message time

A sample of flight track data is shown below:

```
606262,319,TZ,08/08/2005,083054,E,LN70LG,846,260,150,6106N,14817W
601555,319,TZ,08/08/2005,083054,E,LN72LG,772,440,340,5854N,14053W
599139,319,TZ,08/08/2005,083059,E,ASA173,604,450,360,5813N,13701W
```

Flight Arrival (AZ) File

The following fields are included in the flight arrival file:

Data Field	Description
FID	Unique identifier
Rule No	Internal (Flight Explorer) rule no.
Message Type	RZ
Message Time	GMT message was sent
Center	FAA Air Route Traffic Control Center (ARTCC) generating the message
Aircraft ID	Aircraft identifier
CID	FAA computer identifier
Origin	Departure airport
Destination	Arrival airport
Arrival Time	Time of aircraft arrival

A sample of flight arrival data is shown below:

```
160785235,306,AZ,07/27/2005,000002,5,ACA617,CYHZ,CYYZ,07/27/2005,000000
160805098,306,AZ,07/27/2005,000000,S,ASA347,SMF,SEA,07/26/2005,235800
160802518,306,AZ,07/27/2005,000010,G,MEP9,DEN,MKE,07/27/2005,000000
```

APPENDIX B: DEMAND ANALYSIS SCRIPTS AND DATA

This appendix contains the scripts used in the departure time analysis. The following files are included in this appendix.

Python Script to Create a Summary Listing of Departure Time Prediction Errors for Analysis (departtime6.py)

```
"""
departtime.py reads data (ETMS, ASPM, etc), saves into class data
structure and outputs variation histogram bins
departtime6.py is version 6
- Changed flight id key to Callsign + '_' + Origin Airport + '_' + Depa...
...rture Date
Order:
    - Convective Info
    - Airport time zone (convert OAG to UTC)***
    - Airport taxi time (add to push-back time to get wheels-off time)*...
...**
    - Airport efficiency
    - Analysis days (which ETMS days to include)
    - Departure times (ETMS)
    - ETMS Centers (1 letter to 3 letter identifier conversion)
    - Center crossings
    - Weather impact
    - Cancellations
    - Apt classification (Large, Foreign, etc.)
    - OAG***
    - Flight plan***
    - Amended plan***
"""
"""
Flight status:
a.    scheduled  ETMS created this flight from OAG schedule, but has n...
...ot yet received a flight plan. ETMS expects this flight to operate.
b.    airline_created  ETMS created this flight from airline message, ...
...but has not yet received a flight plan. Flight was not in OAG. ETMS ex...
...pects this flight to operate.
c.    filed  ETMS has received an FZ for this flight. ETMS expects thi...
...s flight to operate.
d.    controlled  Flight has been assigned control departure and arriv...
...al times. Flight may have been created from OAG, airline message, or f...
...light plan. ETMS expects this flight to operate.
e.    active  ETMS has received a NAS message indicating this flight i...
...s airborne, but it has not landed yet.
f.    completed  ETMS has received indication this flight has landed.
g.    cancelled  ETMS does not expect this flight to operate. Flight m...
...ay have been created from OAG, airline message, or flight plan.
    1.    The actual time of departure from a DZ message
        - DZ Table have this.
    2.    An estimated time of departure determined by some other activ...
...ation message
        - Same info as DZ table but may be released earlier to system (...
```

```
...ignore)
    3.    A controlled departure time, unless an airline has provided a...
... later departure time
        - Don't use now. In new ETMS data on server?
    4.    An airline-provided runway departure time
        - Source?
    5.    An airline-provided gate departure time plus modeled ground t...
...ime
        - Source
    6.    A flight-plan proposed departure time plus modeled ground tim...
...e
        - Have this and used
    7.    A scheduled departure time plus modeled ground time
        - Have this and used
"""
import string
import sys
import fileinput
import time
import os
import pdb
from cls5 import ETMSTime, Airport, DepartureTimes, FlightData, Center
from datetime import date, datetime, timedelta
import pytz
from pytz import timezone
from timeutils2 import convert_to_utc, etms_to_utc, utc_round_down_15, ...
...utc_delta_time, day_15_minute_str, atalab_to_utc
from timeutils2 import early_utc_time
# Directories
#data_dir = "C:/Documents and Settings/Jeff/My Documents/Research/Data/...
...Flight Explorer Data/"
data_dir = "C:/Users/Jeff/Data/Flight Explorer Data/"
#data_dir = "/Users/antoniotrani/Documents/Jeff/Flight Explorer Data/"
#output_dir = "D:/Departure Time Variation"
output_dir = "C:/Users/Jeff/Analysis/Departure Time Variation"
#output_dir = "/Users/antoniotrani/Documents/Jeff/Departure Time Variat...
...ion/"
taxi_dir = output_dir + "/2 Data/Taxi Times/"
VMCIMC_dir = output_dir + "/2 Data/Airport IFR VFR/"
atalab_dir = "C:/Users/Jeff/Data/ATALAB ETMS Data/"
#atalab_dir = "C:/Documents and Settings/Jeff/My Documents/Research/Dat...
...a/ATALAB ETMS Data/"
#atalab_dir = "/Users/antoniotrani/Documents/Jeff/ATALAB ETMS Data/"
# Input Files
OAG_dir = output_dir + "/2 Data/FSDS/OAGALLDAYS/"
airport_info = output_dir + "/2 Data/Airport Type/airportinfo.csv"
airport_time_zones = output_dir + "/2 Data/Airport Time Zones/Airport T...
```

```
...ime Zones.csv"
analysis_days = output_dir + "/1 Select Analysis Days/ConvectionSeason....
...txt"
convective_sigmet_info = output_dir + "/2 Data/NAVAID ARTCC/wst_artcc.t...
...xt"
etms_centers_one_letter = output_dir + "/center_identifiers.csv"
# Output Files
dep_error_file = output_dir + "/departuretimeerrors-Convection.txt"
apt_error_file = output_dir + "/airport15minavgerror.txt"
#airport_info = output_dir + "/airportinfo.txt"
missing_airports = output_dir + "/missingairports.txt"
airline_cnt = output_dir + "/airlines.txt"
# Constants
speed_of_sound = 573.5  # At high altitudes, average
# Containers
lookahead = [0, 5, 15, 30, 60, 120]
ts_types = ["AREA_TS", "AREA_SEV_TS", "LINE_TS", "LINE_SEV_TS", "AREA_T...
...S_W_TOPS", "AREA_SEV_TS_W_TOPS", "LINE_TS_W_TOPS", "LINE_SEV_TS_W_TOPS...
..."]
top_25_airlines = ["SWA","DAL","AAL","UAL","USA","NWA","EGF","COA","SKW...
...","BTA","MES","TWA","ACA","CAA","FDX","BLR","COM","AWE","QXE","BAW","U...
...PS","ASA","AMW","PDT","ARN"]
alpha_caps = ['A','B','C','D','E','F','G','H','I','J','K','L','M','N','...
...O','P','Q','R','S','T','U','V','W','X','Y','Z']
airports = {}
icao_ids = {}   # Key is IATA ids
flights = {}
#FID = {}
analysisdays = {}
centers = {}
etms_centers = {}   # One letter ETMS center identifiers
org_apt_filter = ''
des_apt_filter = ''
# Checks to see if flight should be considered based on O-D pair of air...
...port
def in_apt_filter(origin_airport, destination_airport, org_class, des_c...
...lass):
    if inc_all_flights_raw:
        return True

    if (origin_airport not in airports) or (destination_airport not in ...
...airports):
        return False

    if (org_apt_filter != org_class) and \
        (org_apt_filter != origin_airport) and \
            (org_apt_filter != airports[origin_airport].altname) and \
```

```
                (org_apt_filter != 'ALL'):
        return False

    if (des_apt_filter != des_class) and \
        (des_apt_filter != destination_airport) and \
            (des_apt_filter != airports[destination_airport].altname) a...
...nd \
                (des_apt_filter != 'ALL'):
        return False

    return True
# read nav_artcc.py output, nav_artcc.py is a parser for Convective Sig...
...met info
def read_convective_sigmet_info():
    for line in fileinput.input(convective_sigmet_info):
        data = line.split(',')
        date2 = data[0]     # 20050611
        time2 = data[1]     # 110055

        #utcyyyy-mm-dd hh:mm:ss
        datetime2 = date2[0:4] + '-' + date2[4:6] + '-' + date2[6:8] + ...
...' '
        datetime2 = datetime2 + time2[2:4] + ':' + time2[4:6] + ':00'

        cntr = data[2]      # ARTCC (center)
        ts_type = data[3]   # Thunderstorm type
        echo_tps = data[4]  # Echo tops

        if (cntr not in centers):
            centers[cntr] = Center(cntr)

        centers[cntr].addforecast(datetime2, ts_type, echo_tps)



# Read airport time zone
def read_airport_time_zone():
    """
    for line in fileinput.input(airport_time_zones):
        data = line.split(',')
        apt = data[0]
        kapt = 'K' + apt
        tz = data[1].replace(' ', '')

        if apt not in airports:
            airports[apt] = Airport(apt)
            airports[kapt] = Airport(apt)
```

```
        airports[apt].timezone = tz
        airports[kapt].timezone = tz
    """


    #file structure:
    #Region, IATA_Id, ICAO_Id, Time_Zone
    for line in fileinput.input(airport_time_zones):
        data = line.split(',')
        region = data[0].replace(' ','')
        apt_iata = data[1].replace(' ','')
        apt_icao = data[2].replace(' ','')
        tz = data[3].replace(' ','')
        tz = tz.replace('\n','')
        tz = tz.replace('\r','')

        if (region == 'US'):
            if apt_iata not in airports:
                airports[apt_iata] = Airport(apt_iata)
                airports[apt_icao] = Airport(apt_iata)

            airports[apt_iata].timezone = tz
            airports[apt_iata].altname = apt_icao
            airports[apt_icao].timezone = tz
            airports[apt_icao].altname = apt_iata
        else:
            if apt_icao not in airports:
                airports[apt_icao] = Airport(apt_icao)

            airports[apt_icao].timezone = tz
            airports[apt_icao].altname = apt_iata

            icao_ids[apt_iata] = apt_icao


# Read taxiing times
def read_airport_taxi():
    for f in os.listdir(taxi_dir):
        taxi_file = os.path.join(taxi_dir, f)
        csv_ext = ('.csv' in f) or ('.CSV' in f)
        if os.path.isfile(taxi_file) and csv_ext:
            for line in fileinput.input(taxi_file):
                if ('SELLCARRNAME' in line):    # Ignore first line
                    continue

                data = line.split(',')
```

```
                apt = data[0]
                kapt = 'K' + apt
                date_mn1 = data[1]
                avgtxi = float(data[2])
                if apt not in airports:
                    airports[apt] = Airport(apt)
                    airports[kapt] = Airport(kapt)

                date_utc = convert_to_utc(date_mn1, airports[apt].timez...
...one)
                airports[apt].addtaxiinterval(date_utc, avgtxi)
                airports[kapt].addtaxiinterval(date_utc, avgtxi)

# Read airport efficiency
def read_apt_efficiency():
    for f in os.listdir(VMCIMC_dir):
        apt_file = os.path.join(VMCIMC_dir, f)
        csv_ext = ('.csv' in f) or ('.CSV' in f)
        if os.path.isfile(apt_file) and csv_ext:
            for line in fileinput.input(apt_file):
                data = line.split(',')

                if (len(data) < 6):
                    print line
                    continue

                apt = data[0]
                kapt = 'K' + apt
                datetm = data[1]
                date_utc = convert_to_utc(datetm, airports[apt].timezon...
...e)

                airports[apt].addweathercondition(date_utc, data[2])
                airports[kapt].addweathercondition(date_utc, data[2])

                airports[apt].adddemand(date_utc, data[3], data[3], dat...
...a[4])
                airports[kapt].adddemand(date_utc, data[3], data[3], da...
...ta[4])

    #print date_utc in airports['IAH'].weather
    #print airports['IAH'].weather[date_utc]


# Read file to determine which ETMS days to include
def read_analysis_days():
    for line in fileinput.input(analysis_days):
```

```
        data = line.split(' ')
        analysisdays[data[1].replace('\n', '')] = data[0]
# Read airport classification
def read_apt_classification():
    for line in fileinput.input(airport_info):
        data = line.split(',')
        aptname1 = data[0].replace(' ', '')
        aptname2 = data[1].replace(' ', '')
        classify = data[2].replace(' ', '')
        classify = classify.replace('\n', '')
        classify = classify.replace('\r', '')
        if (aptname1 not in airports):
            airports[aptname1] = Airport(aptname1)
        airports[aptname1].altname = aptname2
        airports[aptname1].classification = classify

        if (aptname2 not in airports):
            airports[aptname2] = Airport(aptname2)
        airports[aptname2].altname = aptname1
        airports[aptname2].classification = classify

def get_apt_classification(apt):
    if apt in airports:
        return airports[apt].classification
    else:
        kapt = 'K' + apt
        if kapt in airports:
            return airports[kapt].classification
        else:
            return ''

# Read departure times
def read_dep_times():
    airlines = {}
    for f in analysisdays:
        dzname = data_dir + analysisdays[f] + '/dz' + f + '.csv'
        print dzname
        if os.path.isfile(dzname):
            for line in fileinput.input(dzname):
                data = line.split(',')

                message_day = data[3]
                message_time = data[4]
                message_datetime = etms_to_utc(message_day, message_tim...
...e)
                message_datetime15 = utc_round_down_15(message_datetime...
...)
```

```
                dep_day = data[13]
                dep_time = data[14]
                dep_datetime = etms_to_utc(dep_day, dep_time)
                dep_datetime15 = utc_round_down_15(dep_datetime)

                #flightid = data[6] + "_" + data[0] + "_" + dep_day
                orgapt = data[11]
                desapt = data[15]
                orghubsize = get_apt_classification(orgapt)
                deshubsize = get_apt_classification(desapt)
                callsign = data[6]
                flightid = callsign + '_' + orgapt + '_' + dep_day

                if (not in_apt_filter(orgapt,desapt,orghubsize,deshubsi...
...ze)):
                    continue

                tmp_fid = data[6]
                tmp_fid = tmp_fid[0:3]
                if (tmp_fid not in airlines):
                    airlines[tmp_fid] = 1
                else:
                    airlines[tmp_fid] = airlines[tmp_fid] + 1

                if(message_day != dep_day) or (flightid in flights):   ...
... #Only include the first filed dep time (int'l flights)
                    #print message_day + " -- " + dep_day
                    continue
                else:
                    fl = FlightData(flightid)
                    fl.orgapt = orgapt
                    fl.desapt = desapt
                    fl.lvehub = orghubsize
                    fl.arrhub = deshubsize
                    fl.actualwheelsoff = dep_datetime

                    if (orgapt in airports) and (dep_datetime15 in airp...
...orts[orgapt].weather):
                        fl.taxitime = airports[orgapt].getavgtaxitime(d...
...ep_datetime15)
                        last_taxi_time = utc_delta_time(dep_datetime15,...
... 0, -15, 0)
                        last_taxi = airports[orgapt].getavgtaxitime(las...
...t_taxi_time)
                        if (fl.taxitime > last_taxi):
                            fl.taxirate = 'Increasing'
```

```
                        else:
                            fl.taxirate = 'Decreasing'

                        fl.depaptweather = airports[orgapt].weather[dep...
..._datetime15]
                        fl.deptotaldemand = airports[orgapt].totaldeman...
...d[dep_datetime15]
                        fl.depdepartures = airports[orgapt].departures[...
...dep_datetime15]
                        fl.deputilization = airports[orgapt].capacityut...
...il[dep_datetime15]


                        #efficiency_key = airports[orgapt].getefficienc...
...ykey(fl.actualwheelsoff)

                        #if (efficiency_key != None):
                        #    fl.caputil = airports[orgapt].capacityutil...
...ized[efficiency_key]
                        #    fl.depeff = airports[orgapt].depefficiency...
...[efficiency_key]

                    else:    #Use generic ETMS 15 minutes
                        fl.taxitime = 15

                    if (desapt in airports) and (dep_datetime15 in airp...
...orts[desapt].weather):
                        fl.arraptweather = airports[desapt].weather[dep...
..._datetime15]
                        fl.arrtotaldemand = airports[desapt].totaldeman...
...d[dep_datetime15]
                        fl.arrutilization = airports[desapt].capacityut...
...il[dep_datetime15]

                        #efficiency_key2 = airports[desapt].getefficien...
...cykey(fl.actualwheelsoff)
                        #if (efficiency_key2 != None):
                        #    fl.arreff = airports[desapt].arrefficiency...
...[efficiency_key2]

                    flights[flightid] = fl
                    #FID[data[6]+orgapt+dep_day] = data[0]   # Check if...
... FID needs +date key

    w = open(airline_cnt, 'w')
    for al in airlines:
        w.write(al + ',' + str(airlines[al]) + '\n')
```

```
    w.close

# Reads a file containing "1 letter center id", "3 letter center id"
# and stores this information into the etms_centers data structure
def read_etms_centers():
    for line in fileinput.input(etms_centers_one_letter):
        data = line.split(',')
        letter2 = data[0]
        center2 = data[1].replace('\n', '')
        center2 = center2.replace('\r', '')
        etms_centers[letter2] = center2

    #print etms_centers
# Parses the UZ file to obtain the centers and center entry times
def read_center_crossings():
    for f in analysisdays:
        uzname = data_dir + analysisdays[f] + '/uz' + f + '.csv'
        print uzname
        if os.path.isfile(uzname):
            for line in fileinput.input(uzname):
                data = line.split(',')
                callsign = data[6]
                depday = data[3]
                orgapt = data[16].split('.')[0]
                flightid = callsign + '_' + orgapt + '_' + depday
                #flightid = data[6] + "_" + data[0] + "_" + data[3]

                center_1 = data[5]  # One letter center id
                if (center_1 in etms_centers):
                    center_3 = etms_centers[center_1]    # Three letter...
... center id

                    crossing_time = etms_to_utc(data[3], data[4])
                    crossing_time_15 = utc_round_down_15(crossing_time)...
...
                    if (flightid in flights):
                        flights[flightid].centerentrytime[center_3] = c...
...rossing_time_15
# Matches the center crossing and convective sigment info
def enroute_weather_impact():
    for fl in flights:

        for typ in ts_types:
            flights[fl].thunderstorms[typ] = 0

        for ctr in flights[fl].centerentrytime:
            flt_tm = flights[fl].centerentrytime[ctr]
```

```
            if ctr not in centers:
                continue

            if (flt_tm in centers[ctr].severity):
                ts_sev = centers[ctr].severity[flt_tm]
                ts_echo_tops = centers[ctr].echotops[flt_tm]
                flights[fl].thunderstorms[ts_sev] = flights[fl].thunder...
...storms[ts_sev] + 1

                #print " "
                #print "Altitude: " + str(flights[fl].altitude)
                #print "Echo tops: " + str(ts_echo_tops)
                #print " "

                if (flights[fl].altitude == ''):
                    flights[fl].thunderstorms[ts_sev + "_W_TOPS"] = fli...
...ghts[fl].thunderstorms[ts_sev + "_W_TOPS"] + 1
                elif (int(flights[fl].altitude.replace('FL', '')) <= in...
...t(ts_echo_tops.replace('FL', ''))):
                    flights[fl].thunderstorms[ts_sev + "_W_TOPS"] = fli...
...ghts[fl].thunderstorms[ts_sev + "_W_TOPS"] + 1

    #print str("N7924Q_62853698_05/10/2000" in flights)
# Read cancellation messages
def read_cancellations():
    for f in analysisdays:
        rzname = data_dir + analysisdays[f] + '/rz' + f + '.csv'
        print rzname
        if os.path.isfile(rzname):
            for line in fileinput.input(rzname):
                data = line.split(',')
                callsign = data[6]
                orgapt = data[8]
                depday = data[3]
                flightid = callsign + '_' + orgapt + '_' + depday
                #flightid = data[6] + "_" + data[0] + "_" + data[3]
                if (flightid in flights):
                    flights[flightid].cancelled = 1
                #else:
                #    flights[flightid].cancelled = 0


# Read Official Airline Guide (OAG) data (Source:FSDS)
def read_OAG():
    n=0
    for f in os.listdir(OAG_dir):
```

```
        fname = os.path.join(OAG_dir, f)
        if os.path.isfile(fname):
            for line in fileinput.input(fname):
                if ('SELLCARR' in line):
                    continue

                data = line.split('\",\"')
                if (len(data) < 61):
                    print f
                    print line
                    continue

                operator_id = data[2].replace('\"', '')
                fltno = data[4].replace('\"','')
                fltno = fltno.lstrip('0')
                flightid = operator_id+fltno

                # Leaving/Arriving airport (Foreign, Small, Medium, Lar...
...ge, Non Hub)
                lhub = data[59].replace(' ', '-')
                ahub = data[60].replace(' ', '-')
                ahub = ahub.replace('\n','')
                ahub = ahub.replace('\r','')
                ahub = ahub.replace('\"', '')
                ahub = ahub.replace(',', '')

                orgapt = data[5]
                if (lhub == 'Foreign') and (orgapt in icao_ids):
                    orgapt = icao_ids[orgapt]
                elif (lhub == 'Foreign') and (orgapt not in icao_ids):
                    continue
                elif (lhub != 'Foreign') and (orgapt not in airports):
                    continue

                time2 = data[7]
                if (len(time2) == 3):
                    time2 = '0' + time2
                date2 = data[11].split('/')
                #dd/mm/yyyy
                if (len(date2[0]) < 2):
                    date2[0] = '0' + date2[0]
                if (len(date2[1]) < 2):
                    date2[1] = '0' + date2[1]

                #yyyy-mm-dd hh:mm:ss
                date2 = date2[2] + '-' + date2[0] + '-' + date2[1]
                date2 = date2 + ' ' + time2[0:2] + ':' + time2[2:4] + '...
```

```
...:00'

                #if (flightid == 'DAL721') and (data[11] == '09/26/2003...
...'):
                #    pdb.set_trace()

                # Convert to UTC time
                #print date2
                if (airports[orgapt].timezone == ''):
                    continue

                date2 = convert_to_utc(date2, airports[orgapt].timezone...
...)

                # Just date
                date3 = date2.split(' ')[0]
                date3 = date3.split('-')
                date3 = date3[1] + '/' + date3[2] + '/' + date3[0]

                flt_key = flightid + '_' + orgapt + '_' + date3
                if (flt_key not in flights):
                    flt_key = flightid + airports[orgapt].altname + dat...
...e3
                    if (flt_key not in flights):
                        #print line
                        continue

                n = n+1

                flightid = flt_key
                #flightid = flightid+"_"+FID[flt_key]+"_"+date3

                # OAG departure time
                messaget = utc_delta_time(date2, -12, 0, 0)     # Use a...
... dummy date 12 hrs before departure
                flights[flightid].addoagdep(messaget, date2)
                flights[flightid].addestimateddep(messaget, date2)
                #lvetime = data[7].replace('\"', '')
                #dptime = 60.0*float(lvetime[0:2]) + float(lvetime[2:4]...
...)
                #flights[flightid].addoagdep(0.0, dptime)
                #flights[flightid].addestimateddep(0.0, lvetime)

                # Available seats
                flights[flightid].availableseats = data[17]

                # Statute miles
```

```
                flights[flightid].statutemiles = data[18].replace(',', ...
...'')

                # Passenger miles
                flights[flightid].paxmiles = data[19].replace(',','')

                # Leaving/Arriving OEP airport
                flights[flightid].lveoep = data[36]
                flights[flightid].arroep = data[37]

                # Leaving/Arriving airport (Foreign, Small, Medium, Lar...
...ge, Non Hub)
                #lhub = data[59].replace(' ', '-')
                #ahub = data[60].replace(' ', '-')
                flights[flightid].lvehub = lhub
                flights[flightid].arrhub = ahub

                # Set airport data
                fl = flights[flightid]
                if (fl.orgapt not in airports):
                    airports[fl.orgapt] = Airport(fl.orgapt)
                airports[fl.orgapt].classification = lhub

                if (fl.desapt not in airports):
                    airports[fl.desapt] = Airport(fl.desapt)
                airports[fl.desapt].classification = ahub


    print "Loaded " + str(n) + " flights from OAG data."


# Read flight plan file
def read_flight_plan():

    #print flights
    for f in analysisdays:
        fzname = data_dir + analysisdays[f] + '/fz' + f + '.csv'
        print fzname
        if os.path.isfile(fzname):
            for line in fileinput.input(fzname):
            #for line in fileinput.input(FZ_file):
                data = line.split(',')
                message_day = data[3]
                message_time = data[4]
                dep_day = data[14]
                dep_time = data[15]
                callsign = data[6]
```

```
                orgapt = data[12]
                flightid = callsign + '_' + orgapt + '_' + dep_day
                #flightid = data[6] + "_" + data[0] + "_" + dep_day

                if (flightid not in flights) or (message_day != dep_day...
...) or (len(message_time) < 6) or (len(dep_time) < 6):
                    continue

                flt = flights[flightid]

                flt.addestimateddep(etms_to_utc(message_day, message_ti...
...me), etms_to_utc(dep_day, dep_time))
                flt.airspeed = data[11].replace('\t', '')

                #if ("CFHLL" in flt.flightid) or ("N82017" in flt.fligh...
...tid) or ("N515CW" in flt.flightid):
                #    print "\n" + flt.flightid
                #    print data[11] + '\n'

                if ('M' in flt.airspeed):
                    airsp = float(flt.airspeed.replace('M', ''))
                    airsp = airsp * speed_of_sound
                    flt.airspeed = str(int(airsp))
                elif (flt.airspeed[0] in alpha_caps):
                    flt.airspeed = ''


                if(len(data[17]) < 6):  # Should be a time HHMMSS field...
...                     flt.altitude = data[17]
                else:
                    flt.altitude = data[18]

                # Check if different formatted altitude
                for ac in alpha_caps:
                    if ac in flt.altitude:
                        flt.altitude = ''

    print 'End flight plan info...'
    #pdb.set_trace()


# Read amended flight plan file
def read_amended_plan():
    for f in analysisdays:
        afname = data_dir + analysisdays[f] + '/af' + f + '.csv'
        print afname
        if os.path.isfile(afname):
```

```
            for line in fileinput.input(afname):
            #for line in fileinput.input(AF_file):
                data = line.split(',')
                #flightid = data[6] + "_" + data[0]
                message_day = data[3]
                message_time = data[4]
                dep_day = data[16]
                dep_time = data[17]
                callsign = data[6]
                orgapt = data[8]
                flightid = callsign + '_' + orgapt + '_' + dep_day
                #flightid = data[6] + "_" + data[0] + "_" + dep_day

                if (flightid not in flights) or (message_day != dep_day...
...) or (len(message_time) < 6) or (len(dep_time) < 6):
                    continue

                flights[flightid].addestimateddep(etms_to_utc(message_d...
...ay, message_time), etms_to_utc(dep_day, dep_time))

    print 'End amendment info...'
    #pdb.set_trace()
# Reads control departure time from ATALab data
def read_atalab_data():
    num_flights_in_DZ = 0
    num_flights_not_in_DZ = 0

    for f in analysisdays:
        ataname = atalab_dir + "new_flight_20" + f + ".csv"
        print ataname
        if os.path.isfile(ataname):
            for line in fileinput.input(ataname):
                if "ACT_DATE" in line:
                    continue

                data = line.split(',')

                orgapt = data[7]
                if len(orgapt) == 0 or orgapt not in airports:
                    continue

                flightid = data[1]
                dep_day = data[4]
                yyyy = dep_day[0:4]
                mm = dep_day[4:6]
                dd = dep_day[6:8]
                dep_day = mm + '/' + dd + '/' + yyyy
```

```
                #if num_flights_not_in_DZ > 100:
                #    continue

                flt_key = flightid + '_' + orgapt + '_' + dep_day
                #print flt_key
                if (flt_key not in flights):
                    flt_key = flightid + airports[orgapt].altname + dep...
...._day
                    if (flt_key not in flights):
                        num_flights_not_in_DZ = num_flights_not_in_DZ +...
... 1
                        #print line
                        continue

                num_flights_in_DZ = num_flights_in_DZ + 1
                flightid = flt_key
                #flightid = flightid+"_"+FID[flt_key]+"_"+dep_day

                # Check flight flew flag
                flight_flew = data[16]
                if '0' in flight_flew:  # Not enough info to say flight...
... has flown
                    flights[flightid].flew_flag = flight_flew
                    #if flightid in flights:
                    #    del flights[flightid]
                    #continue

                # utc datetime_str in format yyyy-mm-dd hh:mm:ss
                control_dep_time = data[35]
                message_t = data[6]
                message_t2 = atalab_to_utc(message_t)
                fl_dz = flights[flightid].actualwheelsoff #data[25]
                if len(control_dep_time) > 0 and fl_dz == early_utc_tim...
...e(fl_dz, message_t2):
                    ctrl_dep_2 = atalab_to_utc(control_dep_time)
                    #ctrl_dep_2 = control_dep_time[0:4] + '-' + control...
...._dep_time[4:6]
                    #ctrl_dep_2 = ctrl_dep_2 + '-' + control_dep_time[6...
...:8] + ' '
                    #ctrl_dep_2 = ctrl_dep_2 + control_dep_time[9:17]

                    message_t2 = utc_delta_time(ctrl_dep_2, -1, 0, 0)  ...
... # One hour before
                    #message_t2 = message_t[0:4] + '-' + message_t[4:6]...
...                    #message_t2 = message_t2 + '-' + message_t[6:8] + '...
... '
```

```
                    #message_t2 = message_t2 + message_t[9:16]

                    #Wheels off time
                    flights[flightid].addcontroltime(message_t2, ctrl_d...
...ep_2)

                #continue
                # Flight schedule
                FS_time = data[31]
                if len(FS_time) > 0:
                    #pdb.set_trace()
                    FS_time2 = atalab_to_utc(FS_time)
                    message_t2 = utc_delta_time(FS_time2, -6, 0, 0) # 6...
... Hours before, prevents next day messages (mostly)

                    #message_t = data[4]
                    #message_t2 = atalab_to_utc(message_t)

                    flights[flightid].oag = FS_time2
                    flights[flightid].addestimateddep(message_t2, FS_ti...
...me2)

    #print "Number of Flights in DZ: " + str(num_flights_in_DZ)
    #print "Number of Flights not in DZ: " + str(num_flights_not_in_DZ)...
...
    print 'End atalab info...'
    #pdb.set_trace()
# Remove flights that didn't fly (according to ETMS flight flew flag)
def remove_flights_not_flown():
    flights2 = flights.keys()
    for fl in flights2:
        if '0' in flights[fl].flew_flag:
            del flights[fl]
# Print flight info
def print_flight_info(fl_info):
    fl = flights[fl_info]
    print fl.flightid
    print fl.orgapt
    for dt in fl.estimateddeptimes:
        print str(dt.timeofmessage) + " " + str(dt.pushbacktime)
    print fl.actualwheelsoff
    print fl.taxitime
# Calculate statistics for look ahead times
def calculate_bins():
    w=open(dep_error_file,'w')
    header = "Hub Type, Flight, 0minLA, 5minLA, 15minLA, 30minLA, 60min...
...LA, 120minLA, "
```

```
    header = header + "Origin Airport, Destination Airport, "
    #header = header + "Available Seats, Statute Miles, "
    #header = header + "Pax Miles, "
    #header = header + "Scheduled Dep Time, Wheels Off Time, "
    header = header + "Airspeed, "
    header = header + "Altitude, Number of Amendments, "
    header = header + "Number of Amendments_EQ_0, "
    header = header + "Number of Amendments_EQ_1, "
    header = header + "Number of Amendments_EQ_2, "
    header = header + "Number of Amendments_EQ_3, "
    header = header + "Number of Amendments_EQ_4, "
    header = header + "Number of Amendments_GEQ_5, "
    header = header + "Last Info Time, "
    header = header + "Departing Hub Size, "
    header = header + "Departing Hub Size EQ Non Hub, "
    header = header + "Departing Hub Size EQ Small, "
    header = header + "Departing Hub Size EQ Medium, "
    header = header + "Departing Hub Size EQ Large, "
    header = header + "Departing Hub Size EQ Foreign, "
    header = header + "Arriving Hub Size, "
    header = header + "Cancelled, Taxi Time, Taxi Rate Change, "
    header = header + "Taxi_Rate_NoInfo, Taxi_Rate_Increasing, Taxi_Rat...
...e_Decreasing, "
    header = header + "Origin Airport Weather, Origin_Apt_VMC, Origin_A...
...pt_IMC, "
    header = header + "Destination Airport Weather, Dest_Apt_VMC, Dest_...
...Apt_IMC, "
    header = header + "Origin Airport Total Demand, "
    header = header + "Destination Airport Total Demand, "
    header = header + "Origin Airport Departures Only, "
    header = header + "Origin Airport Capacity Utilization, "
    header = header + "Destination Airport Capacity Utilization, "
    header = header + "In GDP, "
    header = header + "AREA_TS, AREA_SEV_TS, LINE_TS, LINE_SEV_TS, "
    header = header + "AREA_TS_W_TOPS, AREA_SEV_TS_W_TOPS, LINE_TS_W_TO...
...PS, LINE_SEV_TS_W_TOPS, "
    header = header + "AIRLINE (TOP 25), Wheels Off Time\n"
    w.write(header)

    sort_key_container = {}

    for fl in flights:
        flt = flights[fl]

        #if 'DAL1288_ATL_06/11/2005' in fl:
        #    pdb.set_trace()
```

```
        errors = flt.departureerror()
        if(len(errors) == 0):
            continue

        # Create a key to sort flights
        sort_key = ''
        if (flt.lveoep == 'Y'):
            sort_key = flt.orgapt + '_'
        else:
            sort_key = flt.lvehub + '_'

        if (flt.arroep == 'Y'):
            sort_key = sort_key + flt.desapt + '_'
        else:
            sort_key = sort_key + flt.arrhub + '_'

        #GA_check = fl[1]p
        #if (GA_check.isdigit()):
        #    sort_key = sort_key + 'OTH'
        #elif (flt.lveoep == 'Y'):
        #    sort_key = sort_key + fl[0:3]
        #else:
        #    sort_key = sort_key + 'OTH'

        if (sort_key not in sort_key_container):
            sort_key_container[sort_key] = 1
        else:
            sort_key_container[sort_key] = sort_key_container[sort_key]...
... + 1

        flt.key = sort_key
        #

        #w.write(fl + ', ')
        flt.addtostrbuffer(fl + ', ')

        for la in lookahead:
            mintime = la
            maxtime = 999.9
            curerror = flt.oagerror()
            for dt in errors:
                if (dt.lookaheadtime >= mintime) and (dt.lookaheadtime ...
...<= maxtime) \
                    and dt.error < curerror:
                    maxtime = dt.lookaheadtime
                    curerror = dt.error
            #w.write(str(curerror) + ',')
```

```
    flt.addtostrbuffer(str(curerror) + ',')

flt.addtostrbuffer(flt.orgapt + ', ')
flt.addtostrbuffer(flt.desapt + ', ')
#flt.addtostrbuffer(flt.availableseats + ', ')
#flt.addtostrbuffer(flt.statutemiles + ', ')
#flt.addtostrbuffer(flt.paxmiles + ', ')
#flt.addtostrbuffer(flt.oag + ', ')
#flt.addtostrbuffer(flt.actualwheelsoff + ', ')
flt.addtostrbuffer(flt.airspeed + ', ')
flt.addtostrbuffer(flt.altitude + ', ')

flt.addtostrbuffer(flt.numamendments() + ', ')
namendments = int(flt.numamendments())
for i in range(0,5):
    if (namendments == i):
        flt.addtostrbuffer('1, ')
    else:
        flt.addtostrbuffer('0, ')

if (namendments >= 5):
    flt.addtostrbuffer('1, ')
else:
    flt.addtostrbuffer('0, ')


flt.addtostrbuffer(flt.timelastinfo() + ', ')

flt.addtostrbuffer(flt.lvehub + ', ')
if (flt.lvehub == "Non-Hub"):
    flt.addtostrbuffer('1, 0, 0, 0, 0, ')
elif (flt.lvehub == "Small"):
    flt.addtostrbuffer('0, 1, 0, 0, 0, ')
elif (flt.lvehub == "Medium"):
    flt.addtostrbuffer('0, 0, 1, 0, 0, ')
elif (flt.lvehub == "Large"):
    flt.addtostrbuffer('0, 0, 0, 1, 0, ')
elif (flt.lvehub == "Foreign"):
    flt.addtostrbuffer('0, 0, 0, 0, 1, ')
else:
    flt.addtostrbuffer('0, 0, 0, 0, 0, ')


flt.addtostrbuffer(flt.arrhub + ', ')
flt.addtostrbuffer(str(flt.cancelled) + ', ')
flt.addtostrbuffer(str(flt.taxitime) + ', ')
flt.addtostrbuffer(flt.taxirate + ', ')
```

```
if flt.taxirate == 'No_Info':
    flt.addtostrbuffer('1, 0, 0, ')
elif flt.taxirate == 'Increasing':
    flt.addtostrbuffer('0, 1, 0, ')
elif flt.taxirate == 'Decreasing':
    flt.addtostrbuffer('0, 0, 1, ')
else:
    flt.addtostrbuffer('0, 0, 0, ')

flt.addtostrbuffer(flt.depaptweather + ', ')
if flt.depaptweather == 'VMC':
    flt.addtostrbuffer('1, 0, ')
elif flt.depaptweather == 'IMC':
    flt.addtostrbuffer('0, 1, ')
else:
    flt.addtostrbuffer('0, 0, ')
flt.addtostrbuffer(flt.arraptweather + ', ')
if flt.arraptweather == 'VMC':
    flt.addtostrbuffer('1, 0, ')
elif flt.arraptweather == 'IMC':
    flt.addtostrbuffer('0, 1, ')
else:
    flt.addtostrbuffer('0, 0, ')
flt.addtostrbuffer(flt.deptotaldemand + ', ')
flt.addtostrbuffer(flt.arrtotaldemand + ', ')
flt.addtostrbuffer(flt.depdepartures + ', ')
flt.addtostrbuffer(flt.deputilization + ', ')
flt.addtostrbuffer(flt.arrutilization + ', ')
flt.addtostrbuffer(str(flt.ingdp()) + ', ')

for typ in ts_types:
    #print " "
    #print flt.flightid
    #print flt.thunderstorms
    #print " "
    flt.addtostrbuffer(str(flt.thunderstorms[typ]) + ', ')

aline = fl[0:3]
if aline in top_25_airlines:
    flt.addtostrbuffer(aline + ', ')
else:
    flt.addtostrbuffer('OTH, ')

flt.addtostrbuffer(str(flt.actualwheelsoff))

flt.addtostrbuffer('\n')
```

```
        w.write(flt.key + ', ' + flt.tostring())
        #w.write(flt.orgapt + ', ')
        #w.write(flt.desapt + ', ')
        #w.write(flt.availableseats + ', ')
        #w.write(flt.statutemiles + ', ')
        #w.write(flt.paxmiles + ', ')
        #w.write(str(flt.oag) + ', ')
        #w.write(str(flt.actualwheelsoff) + ', ')
        #w.write(flt.airspeed + ', ')
        #w.write(flt.altitude + ', ')
        #w.write(flt.numamendments() + ', ')
        #w.write(flt.timelastinfo() + ', ')
        #w.write(flt.lvehub + ', ')
        #w.write(flt.arrhub + ', ')
        #w.write(str(flt.cancelled) + ', ')
        #w.write(flt.taxirate + ', ')
        #w.write(flt.caputil + ', ')
        #w.write(flt.depeff + ', ')
        #w.write(flt.arreff + ', ')
        #w.write('\n')
    w.close()

#def write_airport_info():   #Airport Name 1, Airport Name 2, Classific...
...ation
#    w=open(airport_info,'w')
#    for apt in airports:
#        ap = airports[apt]
#        if (ap.classification != ''):
#            if(ap.classification == 'Large' and apt[0] == 'K'):
#                w.write(ap.apt + ', ' + apt[1:len(apt)] + ', ' + ap.cl...
...assification + '\n')
#            elif(ap.classification == 'Large'):
#                w.write(ap.apt + ', K' + ap.apt + ', ' + ap.classifica...
...tion + '\n')
#            else:
#                w.write(ap.apt + ', ' + ap.apt + ', ' + ap.classificat...
...ion + '\n')
#
#
#    for line in fileinput.input(USAirports):
#        data = line.replace('\n', '')
#        apt = data.replace(' ', '')
#        if apt in airports:
#            if (airports[apt].classification == ''):
#                w.write(apt + ', ' + apt + ', Non-Hub\n')
#        else:
#            w.write(apt + ', ' + apt + ', Non-Hub\n')
```

```
#
#
#    w.close()

def calculate_sas_bins():
    if not inc_all_flights_raw:
        return

    w = open(apt_error_file, 'w')

    # Header
    w.write("DATE,TIME,TMIDX," + org_apt_filter + ',' + des_apt_filter ...
...+ ',')
    w.write(org_apt_filter + '-' + des_apt_filter + ",FID,")
    for i in range(0, len(top_25_airlines)):
        w.write(top_25_airlines[i])
        if i < len(top_25_airlines)-1:
            w.write(',')
    w.write('\n')

    # Origin airport
    org_apt_toterror0 = {}   # Total errors (summation) for all flights...
... leaving during time period
    org_apt_errorcnt0 = {}   # Count of flights leaving during time per...
...iod
    #org_apt_average0 = {}

    # Destination airport
    des_apt_toterror0 = {}   # Total errors (summation) for all flights...
... leaving during time period
    des_apt_errorcnt0 = {}   # Count of flights leaving during time per...
...iod
    #des_apt_average0 = {}

    # Origin -> Destination airport flights
    org_des_toterror0 = {}
    org_des_errorcnt0 = {}
    org_des_flt = {}

    # Get calibration days and append dates
    datetime_range_strings = []
    for ky in analysisdays:
        datetime_range_strings = datetime_range_strings+day_15_minute_s...
...tr(analysisdays[ky], ky)

    # Initialize counters and error totals
    for dt in datetime_range_strings:
```

```
        org_apt_toterror0[dt] = 0.0
        org_apt_errorcnt0[dt] = 0
        des_apt_toterror0[dt] = 0.0
        des_apt_errorcnt0[dt] = 0
        org_des_toterror0[dt] = 0.0
        org_des_errorcnt0[dt] = 0

    # Add flight departure time error
    for flt in flights:
        fl = flights[flt]
        if (fl.orgapt == org_apt_filter):
            dt0 = utc_round_down_15(fl.actualwheelsoff)
            de = fl.departureerror()
            if (len(de) == 0):
                continue
            org_apt_toterror0[dt0] = org_apt_toterror0[dt0] + de[0].err...
...or
            org_apt_errorcnt0[dt0] = org_apt_errorcnt0[dt0] + 1

        if (fl.desapt == des_apt_filter):
            dt0 = utc_round_down_15(fl.actualwheelsoff)
            de = fl.departureerror()
            if (len(de) == 0):
                continue
            des_apt_toterror0[dt0] = des_apt_toterror0[dt0] + de[0].err...
...or
            des_apt_errorcnt0[dt0] = des_apt_errorcnt0[dt0] + 1

        if (fl.orgapt == org_apt_filter) and (fl.desapt == des_apt_filt...
...er):
            dt0 = utc_round_down_15(fl.actualwheelsoff)
            de = fl.departureerror()
            if (len(de) == 0):
                continue
            org_des_toterror0[dt0] = org_des_toterror0[dt0] + de[0].err...
...or
            org_des_errorcnt0[dt0] = org_des_errorcnt0[dt0] + 1
            org_des_flt[dt0] = fl.flightid


    # Print averages
    j=0
    datetime_range_strings.sort()
    for i in range(0, len(datetime_range_strings)):
        dt_i = datetime_range_strings[i]
        buffer = str(dt_i).replace(' ',',') + ','
        buffer = buffer + str(j) + ','
```

```
        j=j+1
        if (org_apt_errorcnt0[dt_i] > 0):
            buffer = buffer + str(org_apt_toterror0[dt_i]/org_apt_error...
...cnt0[dt_i]) + ','
        else:
            buffer = buffer + '0.0,'

        if (des_apt_errorcnt0[dt_i] > 0):
            buffer = buffer + str(des_apt_toterror0[dt_i]/des_apt_error...
...cnt0[dt_i]) + ','
        else:
            buffer = buffer + '0.0'

        if (org_des_errorcnt0[dt_i] > 0):
            buffer = buffer + str(org_des_toterror0[dt_i]/org_des_error...
...cnt0[dt_i])
            buffer = buffer + ',' + org_des_flt[dt_i] + ','
            aline = org_des_flt[dt_i][0:3]
            for k in range(0, len(top_25_airlines)):
                if aline == top_25_airlines[k]:
                    buffer = buffer + '1'
                else:
                    buffer = buffer + '0'

                if k < len(top_25_airlines)-1:
                    buffer = buffer + ','
        else:
            for k in range(0,len(top_25_airlines)):
                buffer = buffer + ','
        #else:
        #    buffer = buffer #+ '0.0'

        buffer = buffer + '\n'

        #print buffer
        w.write(buffer)

    #print datetime_range_strings

    # sort with keys: a.sort(key=str.lower)

    w.close()

# Main
print '================================='
print 'Departure Time Variation Analysis'
print '================================='
```

```
t1=time.time()
# If true then filter will not be applied to raw output, only airport_1...
...5min_avg
# If false then filter will be applied to raw output and airport_15min_...
...avg is meaningless
include_all_flights_raw = raw_input("Include all flights in departure t...
...ime calculations (Y/N)? :")
if include_all_flights_raw == 'Y':
    inc_all_flights_raw = True
else:
    inc_all_flights_raw = False


org_apt_filter = raw_input("Origin airport filter     : ")
des_apt_filter = raw_input("Destination airport filter: ")
read_convective_sigmet_info()
read_airport_time_zone()
read_airport_taxi()
read_apt_efficiency()
read_analysis_days()
read_apt_classification()
read_dep_times()
read_etms_centers()
read_center_crossings()
enroute_weather_impact()
read_cancellations()
#read_OAG()  # Change OAG to UTC time
read_flight_plan()
read_amended_plan()
read_atalab_data()
remove_flights_not_flown()
#print_flight_info('SWA1551_3397638')
calculate_bins()
#write_airport_info()
calculate_sas_bins()
print '\nDone after ' + str(time.time()-t1) + ' seconds.'
#os.system('pause')
```

Python Class Definitions (cls5.py)

```
"""
cls.py contains user defined classes related to ETMS data
cls5.py is version 5
"""
import string
import sys
import fileinput
import pdb
from datetime import date, datetime, timedelta
import pytz
from pytz import timezone
from timeutils2 import convert_to_utc, etms_to_utc, utc_round_down_15, ...
...utc_delta_time
class ETMSTime:
    def __init__(self, s):
        self.timestring = s.replace(':', '')
        self.hour = int(self.timestring[0:2])
        self.min = int(self.timestring[2:4])
        self.sec = int(self.timestring[4:6])
    def decimalhour(self):
        return self.hour + self.min/60.0 + self.sec/3600.0
    def decimalminute(self):
        return self.hour*60.0 + self.min + self.sec/60.0
    def decimalsecond(self):
        return self.hour*3600.0 + self.min*60.0 + self.sec
class Center:
    def __init__(self, s):
        self.zid = s    # Center identifier
        self.severity = {}  # AREA_TS, AREA_SEV_TS, etc by datetime key...
...        self.echotops = {}  # FLXXX by datetime key
    def addforecast(self, arg_datetime, arg_severity, arg_echotops):
        # datetime in format 'yyyy-mm-dd hh:mm:ss'
        datetime2 = utc_round_down_15(arg_datetime)
        for i in range(0,8):
            self.severity[datetime2] = arg_severity
            self.echotops[datetime2] = arg_echotops
            datetime2 = utc_delta_time(datetime2, 0, 15, 0)

class Airport:
    def __init__(self, s):
        self.apt = s
        self.altname = s
        self.classification = ''
        self.timezone = ''

        # Begin taxi variables
```

```
        self.avgtaxi = {}   # Format 2000-01-30 12:00

        # Weather conditions
        self.weather = {}       # IMC/VMC

        # Demand data
        self.totaldemand = {}   # Arrivals + Departures
        self.departures = {}    # Departure demand only
        self.capacityutil = {}  # Percent Capacity Utilized

    def addtaxiinterval(self, datetm, avg):
        self.avgtaxi[datetm] = avg

    def getavgtaxitime(self, departtime):
        if departtime in self.avgtaxi:
            return self.avgtaxi[departtime]
            #if self.avgtaxi[departtime] > 0.0:
            #    return self.avgtaxi[departtime]
            #else:
            #    departtime2 = utc_delta_time(departtime, 0, 15, 0)
            #    if self.avgtaxi[departtime2] > 0.0:
            #        return self.avgtaxi[departtime2]
            #    else:
            #        departtime3 = utc_delta_time(departtime, 0, -15, 0...
...)
            #        if self.avgtaxi[departtime3] > 0.0:
            #            return self.avgtaxi[departtime3]
            #        else:
            #            return 15.0
        else:
            return 15.0

    def addweathercondition(self, datetm, cond):
        self.weather[datetm] = cond

    def adddemand(self, datetm, totaldem, dep, caputil):
        self.totaldemand[datetm] = totaldem
        self.departures[datetm] = dep
        self.capacityutil[datetm] = caputil

class DepartureTimes:
    def __init__(self):
        self.timeofmessage = 0.0
        self.pushbacktime = 0.0
        self.wheelsofftime = 0.0

class EstimationError:
```

```
    def __init__(self):
        self.lookaheadtime = 0.0
        self.error = 0.0

class UTCString:
    def __init__(self,s):
        self.dt_str = s

    def PyTz_Object(self):
        utc_tz = pytz.timezone('UTC')
        s_pytz = self.dt_str.split(' ')
        s_day = s_pytz[0].split('-')
        s_tm = s_pytz[1].split(':')
        pytz_dt = datetime(int(s_day[0]), int(s_day[1]), int(s_day[2]),...
... int(s_tm[0]), int(s_tm[1]), int(s_tm[2]), tzinfo=utc_tz)
        return pytz_dt
class FlightData:
    def __init__(self, s):
        self.flightid = s
        self.orgapt = ''
        self.desapt = ''
        self.estimateddeptimes = []
        self.oag = ''
        self.actualwheelsoff = ''
        self.taxitime = 0.0
        self.taxirate = 'No_Info'  # If taxitime is Increasing, Decreas...
...ing, No Information (not reported)
        self.flew_flag = '2'       # Assume initially that flight flew

        # Misc possible factors
        self.availableseats = ''
        self.statutemiles = ''
        self.paxmiles = ''
        self.airspeed = ''
        self.altitude = ''
        self.amendments = 0
        self.lveoep = ''
        self.arroep = ''
        self.lvehub = ''
        self.arrhub = ''
        self.cancelled = 0
        #self.caputil = ''
        #self.depeff = ''
        #self.arreff = ''
        self.depaptweather = ''
        self.arraptweather = ''
        self.deptotaldemand = ''
```

```
        self.arrtotaldemand = ''
        self.depdepartures = ''
        self.deputilization = ''
        self.arrutilization = ''

        # Convective Sigmet Matching Variables
        self.centerentrytime = {}
        self.thunderstorms = {}

        # Control departure times
        self.controltimes = []

        # String representation of flight
        self.str = ''
        self.key = ''

    def addoagdep(self, messaget, pushbackt):
        self.oag = pushbackt
        #self.addestimateddep(messaget, pushbackt)
    def addestimateddep(self, messaget, pushbackt):
        tom = UTCString(messaget)
        tom = tom.PyTz_Object()
        awo = UTCString(self.actualwheelsoff)
        awo = awo.PyTz_Object()

        if awo > tom:
            dt = DepartureTimes()
            dt.timeofmessage = messaget
            dt.pushbacktime = pushbackt
            self.estimateddeptimes.append(dt)
    def addcontroltime(self, messaget, wheelsofftime):
        dt = DepartureTimes()
        dt.timeofmessage = messaget
        dt.wheelsofftime = wheelsofftime
        self.controltimes.append(dt)
        #self.numcontrolamendments = self.numcontrolamendments + 1
    def ingdp(self):
        if len(self.controltimes) > 0:
            return 1
        else:
            return 0
    def departureerror(self):

        deperror = []
        awo = UTCString(self.actualwheelsoff)
        awo = awo.PyTz_Object()
        for dt in self.estimateddeptimes:
```

```
            ee = EstimationError()
            pbt = UTCString(dt.pushbacktime)
            pbt = pbt.PyTz_Object()
            tom = UTCString(dt.timeofmessage)
            tom = tom.PyTz_Object()

            lat = pbt-tom
            ee.lookaheadtime = lat.seconds/60.0

            err = pbt-awo
            if (pbt > awo):
                ee.error = err.days*24.0*60.0 + err.seconds/60.0 - self...
....taxitime
            else:
                ee.error = -1*(err.days*24.0*60.0 + err.seconds/60.0) -...
... self.taxitime

            if pbt>tom:
                deperror.append(ee)

        for dt in self.controltimes:
            ee = EstimationError()
            wot = UTCString(dt.wheelsofftime)
            wot = wot.PyTz_Object()
            tom = UTCString(dt.timeofmessage)
            tom = tom.PyTz_Object()

            lat = wot-tom
            ee.lookaheadtime = lat.seconds/60.0

            err = wot-awo
            if (wot > awo):
                ee.error = err.days*24.0*60.0 + err.seconds/60.0
            else:
                ee.error = -1*(err.days*24.0*60.0 + err.seconds/60.0)

            if wot>tom:
                deperror.append(ee)
        return deperror
    def oagerror(self):
        if (self.oag != ''):
            awo = UTCString(self.actualwheelsoff)
            awo = awo.PyTz_Object()

            pag = UTCString(self.oag)
            pag = pag.PyTz_Object()
```

```
            err1 = awo-pag
            return err1.seconds/60.0-self.taxitime
        else:
            return 999.9
    def numamendments(self):    # Number of flight amendments
        self.amendments = len(self.estimateddeptimes)

        # Don't include control departure times in amendment count
        self.amendments = self.amendments - len(self.controltimes)

        # Airspeed set when filed flight plan read
        if (self.airspeed != ''):
            self.amendments = self.amendments - 1 # -1 for filed flight...
... plan

        if (self.oag != ''):
            self.amendments = self.amendments - 1
        return str(self.amendments)

    def timelastinfo(self):
        mint = 999999.0
        awo = UTCString(self.actualwheelsoff)
        awo = awo.PyTz_Object()
        for dt in self.estimateddeptimes:
            tom = UTCString(dt.timeofmessage)
            tom = tom.PyTz_Object()
            tempt = awo - tom
            tempt = tempt.seconds/60.0
            if (tempt < mint):
                mint = tempt
        return str(mint)
    def clearstrbuffer(self):
        self.str = ''
    def addtostrbuffer(self, s):
        self.str = self.str + s
    def tostring(self):
        return self.str
```

Python Time Utilities (timeutils2.py)

```
"""
timeutils2.py contains functions for converting to/from UTC times and o...
...ther
misc time functions
"""
import string
import sys
import time
from datetime import date, datetime, timedelta
import pytz
from pytz import timezone
import pdb
# Convert local times (ASPM, ASQP, etc.) to UTC time
def convert_to_utc(datetime_str, tzone):
    # datetime_str in format yyyy-mm-dd hh:mm:ss
    data = datetime_str.split(' ')
    date2 = data[0].split('-')
    time2 = data[1].split(':')
    sec2 = '00' # A placeholder for time seconds is appended
    """
    loc_tz = pytz.timezone('UTC')
    if (timezone == 'EST'):
        loc_tz = pytz.timezone('US/Eastern')
    elif (timezone == 'CST'):
        loc_tz = pytz.timezone('US/Central')
    elif (timezone == 'PST'):
        loc_tz = pytz.timezone('US/Pacific')
    elif (timezone == 'MST'):
        loc_tz = pytz.timezone('US/Mountain')
    elif (timezone == 'HAL'):
        loc_tz = pytz.timezone('US/Hawaii')
    elif (timezone == 'AKST'):
        loc_tz = pytz.timezone('US/Alaska')
    elif (timezone == 'PUR'):
        loc_tz = pytz.timezone('Atlantic/Bermuda')
    """

    loc_tz = pytz.timezone(tzone)

    loc_dt = datetime(int(date2[0]), int(date2[1]), int(date2[2]), int(...
...time2[0]), int(time2[1]), 0, tzinfo=loc_tz)
    utc_dt = loc_dt.astimezone(pytz.timezone('UTC'))

    buf = ['','','','']

    if utc_dt.month < 10:
```

```
        buf[0] = '0'

    if utc_dt.day < 10:
        buf[1] = '0'

    if utc_dt.hour < 10:
        buf[2] = '0'

    if utc_dt.minute < 10:
        buf[3] = '0'

    buffer = str(utc_dt.year) + '-' + buf[0] + str(utc_dt.month) + '-' ...
...+ buf[1] + str(utc_dt.day)
    buffer = buffer + ' ' + buf[2] + str(utc_dt.hour) + ':' + buf[3] + ...
...str(utc_dt.minute) + ':' + sec2
    return buffer

# Convert ETMS message format to standard UTC format
def etms_to_utc(etmsdate, etmstime):
    # utc datetime_str in format yyyy-mm-dd hh:mm:ss
    # etmsdate in format mm/dd/yyyy
    # etmstime in format hhmmss
    utc_str = ''
    etmsdate2 = etmsdate.split('/')
    utc_str = etmsdate2[2] + '-' + etmsdate2[0] + '-' + etmsdate2[1] + ...
...' '
    utc_str = utc_str + etmstime[0:2] + ':' + etmstime[2:4] + ':' + etm...
...stime[4:6]
    return utc_str
# Round UTC time down to 15 minute intervals (to use APO data)
# Only only the minutes and seconds will change
def utc_round_down_15(utc_time):
    # utc datetime_str in format yyyy-mm-dd hh:mm:ss
    utc_time2 = utc_time.split(' ')
    date2 = utc_time2[0]
    time2 = utc_time2[1].split(':')
    hour2 = time2[0]
    min2 = int(time2[1])
    sec2 = '00' #time2[2]
    mn = ''
    if min2 < 15:
        mn = '00'
    elif min2 < 30:
        mn = '15'
    elif min2 < 45:
        mn = '30'
    else:
```

```
        mn = '45'
    buffer = date2 + ' ' + hour2 + ':' + mn + ':' + sec2
    return buffer
# Returns a string based on a specified time offset
def utc_delta_time(utc_time, delta_hour, delta_min, delta_sec):
    utc_time2 = utc_time.split(' ')
    date2 = utc_time2[0].split('-')
    time2 = utc_time2[1].split(':')
    utc_tz = pytz.timezone('UTC')
    utc_dt2 = datetime(int(date2[0]), int(date2[1]), int(date2[2]), int...
...(time2[0]), int(time2[1]), int(time2[2]), tzinfo=utc_tz)
    utc_dt_delta = utc_dt2 + timedelta(hours = delta_hour) + timedelta(...
...minutes = delta_min) + timedelta(seconds = delta_sec)
    buf = ['','','','','']

    if utc_dt_delta.month < 10:
        buf[0] = '0'

    if utc_dt_delta.day < 10:
        buf[1] = '0'

    if utc_dt_delta.hour < 10:
        buf[2] = '0'

    if utc_dt_delta.minute < 10:
        buf[3] = '0'
    if utc_dt_delta.second < 10:
        buf[4] = '0'

    buffer = str(utc_dt_delta.year) + '-' + buf[0] + str(utc_dt_delta.m...
...onth) + '-' + buf[1] + str(utc_dt_delta.day)
    buffer = buffer + ' ' + buf[2] + str(utc_dt_delta.hour) + ':' + buf...
...[3] + str(utc_dt_delta.minute) + ':'
    buffer = buffer + buf[4] + str(utc_dt_delta.second)
    return buffer
# Returns an array of utc datetime strings at 15 minute intervals
# for the date specified.
def day_15_minute_str(year1, day1):
    # Format of year1 is yyyy
    # Format of day1 is yymmdd

    # utc datetime_str in format yyyy-mm-dd hh:mm:ss
    utc_str_0 = year1 + '-' + day1[2:4] + '-' + day1[4:6] + ' 00:00:00'...
...    return_utc_str_arr = [utc_str_0]

    # Create the array
    for i in range(0, 4*24-1):  # 4*24-1 creates 15 minute intervals fo...
```

```
...r entire day
        utc_str_0 = utc_delta_time(utc_str_0, 0, 15, 0)
        return_utc_str_arr.append(utc_str_0)

    return return_utc_str_arr
# Converts atalab format (yyyymmdd:hh:mm:ss) to utc time string (yyyy-m...
...m-dd hh:mm:ss)
def atalab_to_utc(time_str):
    time_str2 = time_str.replace(':', '')
    time_str3 = time_str2[0:4] + '-' + time_str2[4:6]
    time_str3 = time_str3 + '-' + time_str2[6:8] + ' '
    time_str3 = time_str3 + time_str2[8:10] + ':'
    time_str3 = time_str3 + time_str2[10:12] + ':'
    time_str3 = time_str3 + time_str2[12:14]

    return time_str3

# Returns the earlier of two UTC strings
def early_utc_time(time1, time2):
    time1a = time1.split(' ')
    time2a = time2.split(' ')

    time1d = time1a[0].split('-')
    time1t = time1a[1].split(':')

    time2d = time2a[0].split('-')
    time2t = time2a[1].split(':')

    for i in range(0,3):
        if int(time1d[i]) < int(time2d[i]):
            return time1
        elif int(time1d[i]) > int(time2d[i]):
            return time2

    for j in range(0,3):
        if int(time1t[j]) < int(time2t[j]):
            return time1
        elif int(time1t[j]) > int(time2t[j]):
            return time2

    return time2
```

Description of Data File Created with Python Scripts

The following fields are included in the departure time error file:

Data Field	Description
Hub Type	Airport hub type for both the origin and destination airport for this flight (Large Hub, Medium Hub, Small Hub, Non-Hub, Foreign)
Flight	Flight callsign identifier
0minLA	Departure time prediction error at a 0 minute look-ahead time to departure (minutes)
5minLA	Departure time prediction error at a 5 minute look-ahead time to departure (minutes)
15minLA	Departure time prediction error at a 15 minute look-ahead time to departure (minutes)
30minLA	Departure time prediction error at a 30 minute look-ahead time to departure (minutes)
60minLA	Departure time prediction error at a 60 minute look-ahead time to departure (minutes)
120minLA	Departure time prediction error at a 120 minute look-ahead time to departure (minutes)
Origin Airport	FAA 3 letter identifier code for the origin airport for this flight
Destination Airport	FAA 3 letter identifier code for the destination airport for this flight
Airspeed	Planned airspeed for this flight (knots)
Altitude	Planned cruising altitude (flight level) for this flight (100's feet)
Number of Amendments	Number of amendment messages transmitted through ETMS for this flight
Number of Amendments EQ 0	Is 1 if there were no amendments, 0 otherwise
Number of Amendments EQ 1	Is 1 if there were 1 amendment, 0 otherwise
Number of Amendments EQ 2	Is 1 if there were 2 amendments, 0 otherwise
Number of Amendments EQ 3	Is 1 if there were 3 amendments, 0 otherwise
Number of Amendments EQ 4	Is 1 if there were 4 amendments, 0 otherwise
Number of Amendments EQ 5	Is 1 if there were 5 amendments, 0 otherwise
Last Info Time	Time since last information about this flight was transmitted through ETMS (minutes)
Departing Hub Size	Hub size for the departure airport
Departing Hub Size EQ Non Hub	Is 1 if the departing airport is a non-hub, 0 otherwise
Departing Hub Size EQ Small	Is 1 if the departing airport is a small hub, 0 otherwise
Departing Hub Size EQ Medium	Is 1 if the departing airport is a medium hub, 0 otherwise
Departing Hub Size EQ Large	Is 1 if the departing airport is a large hub, 0 otherwise

Departing Hub Size EQ Foreign	Is 1 if the departing airport is foreign (outside the U.S.), 0 otherwise
Arriving Hub Size	Hub size for the arriving airport
Cancelled	Is 1 if this flight is cancelled and reactivated, 0 otherwise
Taxi Time	The average taxi time for flights departing the airport within 15 minutes of the flights scheduled departure. This is used to calculate an estimated wheels-off time since all estimated departure times in ETMS are gate push-back times.
Taxi Rate Change	Rate of change of the taxi time by 15 minute time period
Taxi Rate NoInfo	Is 1 if there is no taxi info for this flight, 0 otherwise
Taxi Rate Increasing	Is 1 if the taxi time is increasing from the previous 15 minute time period, 0 otherwise
Taxi Rate Decreasing	Is 1 if the taxi time is decreasing from the previous 15 minute time period, 0 otherwise
Origin Airport Weather	The meteorological conditions at the origin airport, either visual (VMC) or instrument (IMC)
Origin Apt VMC	Is 1 if the origin airport is operating under Visual Meteorological Conditions (VMC), 0 otherwise
Origin Apt IMC	Is 1 if the origin airport is operating under Instrument Meteorological Conditions (IMC), 0 otherwise
Destination Airport Weather	The meteorological conditions at the destination airport, either visual (VMS) or instrument (IMC)
Destination Apt VMC	Is 1 if the destination airport is operating under Visual Meteorological Conditions (VMC), 0 otherwise
Destination Apt IMC	Is 1 if the destination airport is operating under Instrument Meteorological Conditions (IMC), 0 otherwise
Origin Airport Total Demand	Total number of flights arriving and departing during the 15 minute time period when the flight is expected to depart
Destination Airport Total Demand	Total number of flights arriving and departing during the 15 minute time period when the flight is expected to arrive
Origin Airport Departures Only	Number of departures from the origin airport during the 15 minute time period when the flight is expected to depart
Origin Airport Capacity Utilization	The percentage of the total capacity utilized at the origin airport
Destination Airport Capacity Utilization	The percentage of the total capacity utilized at the destination airport
In GDP	Is 1 if this flight is in a ground delay program (GDP), 0 otherwise
AREA TS	Is 1 if this flight is forecasted to be impacted by an area type thunderstorm, 0 otherwise
AREA SEV TS	Is 1 if this flight is forecasted to be impacted by a severe area type thunderstorm, 0 otherwise

LINE TS	Is 1 if this flight is forecasted to be impacted by a line type thunderstorm, 0 otherwise
LINE SEV TS	Is 1 if this flight is forecasted to be impacted by a severe line type thunderstorm, 0 otherwise
AREA TS W TOPS	Same as AREA TS but with consideration of the echo tops of the thunderstorm and the ability of the flight to fly over the weather system
AREA SEV TS W TOPS	Same as AREA SEV TS but with consideration of the echo tops of the thunderstorm and the ability of the flight to fly over the weather system
LINE TS W TOPS	Same as LINE TS but with consideration of the echo tops of the thunderstorm and the ability of the flight to fly over the weather system
LINE SEV TS W TOPS	Same as LINE SEV TS but with consideration of the echo tops of the thunderstorm and the ability of the flight to fly over the weather system
AIRLINE (TOP 25)	Lists the 3 letter airline identifier if the airline is in the top 25, by operations, uses OTH otherwise
Wheels Off Time	The observed time of flight departure measured as the time when the wheels of the aircraft lift off from the runway

A sample from the data file is shown below:

```
Large_Large_, COA733_IAH_09/26/2002, 16.33,16.33,16.33,16.33,999.9,999.9,IAH
   , SFO, 0484, 310, 0, 1, 0, 0, 0, 0, 0, 77.8166666667, Large, 0, 0, 0, 1,
   0, Large, 0, 19.67, Decreasing, 0, 0, 1, VMC, 1, 0, VMC, 1, 0, 18, 21, 18,
    4, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, COA, 2002-09-26 17:31:00
Non-Hub_Non-Hub_, GLA631_IWD_05/10/2001, -5.0,-5.0,-5.0,-5.0,-5.0,-5.0,IWD,
   IMT, 0278, 090, 0, 1, 0, 0, 0, 0, 0, 64.2333333333, Non-Hub, 1, 0, 0, 0, 0,
   Non-Hub, 0, 15, No_Info, 1, 0, 0, , 0, 0, , 0, 0, , , , , , 0, 0, 0, 0, 0,
   0, 0, 0, 0, OTH, 2001-05-10 16:10:00
Large_Small_, DAL641_ATL_07/27/2000, -0.5,-0.5,-0.5,-0.5,-0.5,-0.5,ATL, GSP,
   0457, 210, 0, 1, 0, 0, 0, 0, 0, 130.166666667, Large, 0, 0, 0, 1, 0, Small,
   0, 11.5, Decreasing, 0, 0, 1, IMC, 0, 1, , 0, 0, 52, , 52, 16, , 0, 0, 0, 0
   , 0, 0, 0, 0, 0, DAL, 2000-07-27 14:16:00
Medium_Large_, DAL404_JAX_07/27/2000, 35.0,35.0,35.0,35.0,35.0,35.0,JAX, ATL,
   0484, 290, 0, 1, 0, 0, 0, 0, 0, 266.966666667, Medium, 0, 0, 1, 0, 0, Large
   , 0, 15, No_Info, 1, 0, 0, , 0, 0, VMC, 1, 0, , 38, , , 30, 0, 0, 0, 0, 0,
   0, 0, 0, 0, DAL, 2000-07-27 22:05:00
Large_Medium_, DAL2003_CVG_09/26/2001, -6.5,-6.5,-6.5,-6.5,-6.5,-6.5,CVG, PBI,
   , , 0, 1, 0, 0, 0, 0, 0, 368.0, Large, 0, 0, 0, 1, 0, Medium, 0, 14.5,
   Increasing, 0, 1, 0, VMC, 1, 0, VMC, 1, 0, 26, 2, 26, 13, 2, 0, 2, 0, 0, 0,
   2, 0, 0, 0, DAL, 2001-09-26 00:48:00
```

SAS Script to Calculate Conditional Variance (conditional_variance.sas)

The following SAS script is used to calibrate the model presented in Table 18.

```
/* Import the data */
PROC IMPORT OUT= WORK.CONDVAR
            DATAFILE= "C:\Documents and Settings\Jeff\My Documents\Drop...
... Box\Analysis\Departure Time Variation\d
eparturetimeerrors-RAW.txt"
            DBMS=CSV REPLACE;
     GETNAMES=YES;
     DATAROW=2;
RUN;
/*PROC IMPORT OUT= WORK.CONDVAR
            DATAFILE= "C:\Documents and Settings\Jeff\My Documents\
Drop Box\Analysis\Departure Time Variation\departuretimeerrors-RAW.txt"...
...
            DBMS=CSV REPLACE;
     GETNAMES=YES;
     DATAROW=2;
RUN;*/
data Condvar2;
     set Work.Condvar;
     where _30minLA > -60 and _30minLA < 120;
     log30minLA = log(_30minLA + 60);
     arrlarge = (_Arriving_Hub_Size = "Large");
     arrmedium = (_Arriving_Hub_Size = "Medium");
     arrsmall = (_Arriving_Hub_Size = "Small");
     arrnonhub = (_Arriving_Hub_Size = "Non-Hub");
     arrforeign = (_Arriving_Hub_Size = "Foreign");
     if(_AIRLINE__TOP_25_ = "OTH") then
          top25 = 0;
     else top25 = 1;
     large25 = top25*_Departing_Hub_Size_EQ_Large;
     medium25 = top25*_Departing_Hub_Size_EQ_Medium;
     small25 = top25*_Departing_Hub_Size_EQ_Small;
     nonhub25 = top25*_Departing_Hub_Size_EQ_Non_Hub;
     foreign25 = top25*_Departing_Hub_Size_EQ_Foreign;
     amend_cancelled = _Number_of_Amendments*_Cancelled;
     imc2 = _Origin_Apt_IMC*_Dest_Apt_IMC;
     largelarge = _Departing_Hub_Size_EQ_Large*arrlarge;
     if(top25) then
          nottop25 = 0;
     else nottop25 = 1;
     small25large = (_Departing_Hub_Size_EQ_Small + _Departing_Hub_Size...
..._EQ_Non_Hub)*
               nottop25*(arrmedium + arrsmall + arrnonhub + arrforeign)...
...;
```

```
     if(_AREA_TS + _AREA_SEV_TS + _LINE_TS + _LINE_SEV_TS + _AREA_TS_W_...
...TOPS +
          _AREA_SEV_TS_W_TOPS + _LINE_TS_W_TOPS + _LINE_SEV_TS_W_TOPS >...
... 0) then
          anyconvection = 1;
     else anyconvection = 0;
run;
proc autoreg data=Work.Condvar2;
     model log30minLA = _Departing_Hub_Size_EQ_Large
                    _Departing_Hub_Size_EQ_Medium
                    _Departing_Hub_Size_EQ_Small
                    _Departing_Hub_Size_EQ_Non_Hub
               /
               normal;
     hetero _Departing_Hub_Size_EQ_Large
               _Departing_Hub_Size_EQ_Medium
               _Departing_Hub_Size_EQ_Small
               _Departing_Hub_Size_EQ_Non_Hub
               _Origin_Apt_IMC
                    large25 medium25 nonhub25 foreign25
                    small25large
                    _Number_of_Amendments
                    anyconvection
                    _Cancelled
                    amend_cancelled
               / LINK=EXP;
run;
proc model data = Work.Condvar2;
     log30minLA = b0 + b1*_Departing_Hub_Size_EQ_Large +
                    b2*_Departing_Hub_Size_EQ_Medium +
                    b3*_Departing_Hub_Size_EQ_Small +
                    b4*_Departing_Hub_Size_EQ_Non_Hub;
     h.log30minLA = exp(a0 + a1*_Departing_Hub_Size_EQ_Large +
               a2*_Departing_Hub_Size_EQ_Medium +
               a3*_Departing_Hub_Size_EQ_Small +
               a4*_Departing_Hub_Size_EQ_Non_Hub +
               a5*_Origin_Apt_IMC +
                    a6*large25 +
a7*medium25 +
a8*nonhub25 +
a9*foreign25 +
                    a10*small25large +
                    a11*_Number_of_Amendments +
                    a12*anyconvection +
                    a13*_Cancelled +
                    a14*amend_cancelled);
     fit log30minLA;
```

```
run;
```

SAS Script to Create Tree Diagram for Clustering (cluster_analysis.sas)

The following SAS script is used to create the tree diagram shown in Figure 19.

```
/* Import the data */
PROC IMPORT OUT= WORK.CLUSTER
            DATAFILE= "C:\Users\Jeff\Analysis\Departure Time Variation\...
...d
eparturetimeerrors-RAW.txt"
            DBMS=CSV REPLACE;
     GETNAMES=YES;
     DATAROW=2;
RUN;
/*PROC IMPORT OUT= WORK.CLUSTER
            DATAFILE= "C:\Documents and Settings\Jeff\My Documents\
Drop Box\Analysis\Departure Time Variation\departuretimeerrors-RAW.txt"...
...
            DBMS=CSV REPLACE;
     GETNAMES=YES;
     DATAROW=2;
RUN;*/
proc sql;
     create table Cluster2 as
          select c._30minLA, (100000000*c._Departing_Hub_Size_EQ_Large ...
...+
               200000000*c._Departing_Hub_Size_EQ_Medium +
               300000000*c._Departing_Hub_Size_EQ_Small +
               400000000*c._Departing_Hub_Size_EQ_Non_Hub +
               500000000*c._Departing_Hub_Size_EQ_Foreign +
               10000000*c._Origin_Apt_VMC + 20000000*c._Origin_Apt_IMC ...
...+
               1000000*c._Dest_Apt_VMC + 2000000*c._Dest_Apt_IMC +
               100000*(c._AREA_SEV_TS_W_TOPS + c._LINE_SEV_TS_W_TOPS + ...
...1) +
               10000*(c._In_GDP + 1) +
               1000*(c._Cancelled + 1) +
               100*((c._Number_of_Amendments = 0)) +
               10*((c._AIRLINE__TOP_25_ = "OTH")+1) +
               1*(c._Arriving_Hub_Size = "Large") +
               2*(c._Arriving_Hub_Size = "Medium") +
               3*(c._Arriving_Hub_Size = "Small") +
               4*(c._Arriving_Hub_Size = "Non-Hub") +
               5*(c._Arriving_Hub_Size = "Foreign")
               ) as Catvar,
               c._Departing_Hub_Size_EQ_Large,
               c._Departing_Hub_Size_EQ_Medium,
               c._Departing_Hub_Size_EQ_Small,
               c._Departing_Hub_Size_EQ_Non_Hub,
```

```
                c._Departing_Hub_Size_EQ_Foreign,
                c._Origin_Apt_VMC,
                c._Origin_Apt_IMC,
                c._Dest_Apt_VMC,
                c._Dest_Apt_IMC,
                c._AREA_SEV_TS_W_TOPS,
                c._LINE_SEV_TS_W_TOPS,
                c._In_GDP,
                c._Cancelled,
                c._Number_of_Amendments,
                (c._AIRLINE__TOP_25_ = "OTH") as nottop25,
                (1*(c._Arriving_Hub_Size = "Large") +
                2*(c._Arriving_Hub_Size = "Medium") +
                3*(c._Arriving_Hub_Size = "Small") +
                4*(c._Arriving_Hub_Size = "Non-Hub") +
                5*(c._Arriving_Hub_Size = "Foreign")) as arrhubsize,
                c._Flight
                from Work.Cluster as c
                     where c._30minLA > -60 and c._30minLA < 120;
proc univariate data=Work.Cluster2 noprint;
     where Catvar >= 100000000;
     class Catvar;
     var _30minLA;
     output out=Catstats
          mean=mean std=std n=n;
run;
proc cluster method=ward data=Work.Catstats outtree=Tree;
     where n>50;
     freq n;
     var mean std;
     id Catvar;
run;
proc tree data=Tree out=NewCluster nclusters=10
             graphics haxis=axis1 horizontal;
      height _rsq_;
      copy mean std ;
       id Catvar;
run;
proc sort data=Work.NewCluster;
     by Cluster Catvar;
run;
proc sql;
     create table Cluster3 as
          select * from
               Work.Cluster2 as c, Work.Newcluster as n
                    where c.Catvar = n.Catvar;
data Cluster4;
```

```
     set Cluster3;
     if arrhubsize=1 then
          arrlarge = 1;
     else arrlarge = 0;
     if arrhubsize=2 then
          arrmedium = 1;
     else arrmedium = 0;
     if arrhubsize=3 then
          arrsmall = 1;
     else arrsmall = 0;
     if arrhubsize=4 then
          arrnonhub = 1;
     else arrnonhub = 0;
     if arrhubsize=5 then
          arrforeign = 1;
     else arrforeign = 0;
run;
proc univariate data=Work.Cluster4;
     where _AREA_SEV_TS_W_TOPS = 1 or _LINE_SEV_TS_W_TOPS = 1;
     var _30minLA;
run;
proc univariate data=Work.Cluster4;
     where CLUSTER = 10;
     var _Departing_Hub_Size_EQ_Large
               _Departing_Hub_Size_EQ_Medium
               _Departing_Hub_Size_EQ_Small
               _Departing_Hub_Size_EQ_Non_Hub
               _Departing_Hub_Size_EQ_Foreign
               _Origin_Apt_VMC
               _Origin_Apt_IMC
               _Dest_Apt_VMC
               _Dest_Apt_IMC
               _AREA_SEV_TS_W_TOPS
               _LINE_SEV_TS_W_TOPS
               _In_GDP
               _Cancelled
               _Number_of_Amendments
               nottop25
               arrlarge arrmedium arrsmall arrnonhub arrforeign;
run;
proc sort data= Work.Cluster4;
     by Cluster;
run;
/* Create the final 10 cluster data set */
data ClusterFinal;
     set Work.Cluster4;
     if(_Number_of_Amendments > 0 or _Cancelled > 0) then
```

```
          do;
               if(_Cancelled > 0) then
                    Cluster10 = 9;
               else
                    Cluster10 = 5;
          end;
     else if(_AREA_SEV_TS_W_TOPS > 0 or _LINE_SEV_TS_W_TOPS > 0) then
          Cluster10 = 7;
     else
          do;
               if(_Departing_Hub_Size_EQ_Large = 1 and nottop25 = 0) th...
...en
                    Cluster10 = 3;
               else if(_Departing_Hub_Size_EQ_Large = 1) then
                    Cluster10 = 4;
               else if(_Departing_Hub_Size_EQ_Medium = 1) then
                    Cluster10 = 2;
               else if(_Departing_Hub_Size_EQ_Small = 1 or _Departing_H...
...ub_Size_EQ_Non_Hub = 1) then
                    do;
                         if(nottop25 = 0) then
                              Cluster10 = 8;
                         else if(arrlarge) then
                              Cluster10 = 10;
                         else
                              Cluster10 = 1;
                    end;
               else
                    Cluster10 = 6;
          end;
          if(arrhubsize = 1) then
               arrhubsize2 = "Large Hub";
          else if(arrhubsize = 2) then
               arrhubsize2 = "Medium Hub";
          else if(arrhubsize = 3) then
               arrhubsize2 = "Small Hub";
          else if(arrhubsize = 4) then
               arrhubsize2 = "Non-Hub";
          else if(arrhubsize = 5) then
               arrhubsize2 = "Foreign";
     len = length(_Flight);
     dt = substr(_Flight,len-9,10);
     if(dt = "08/29/2005") then
          date_in_range = 1;
     else date_in_range = 0;
     idx = index(_Flight,"_");
     newFlightId = substr(_Flight,1,idx-1);
```

```
run;
proc univariate data=Work.ClusterFinal;
     class arrhubsize;
     var _30minLA;
run;
proc sql;
     create table ClusterFinal2 as
          select c._30minLA, c.Cluster10, c.arrhubsize
               from Work.ClusterFinal as c
                    where c.arrhubsize > 0;
proc sort data=Work.ClusterFinal2;
     by Cluster10;
run;
/* Create file for kernel smoothing in MATLAB */
proc sql;
     create table ClusterExport as
          select c._30minLA
               from Work.ClusterFinal as c
                    where c.Cluster10 = 10;
/*PROC EXPORT DATA= WORK.Clusterexport
            OUTFILE= "C:\Users\Jeff\Analysis\Mesoscopic Flow Model\Sect...
...o
r Traversal Time Variation\Traversal Analysis\cluster10.csv"
            DBMS=CSV REPLACE;
RUN;*/
/* Create file listing cluster for each flight */
proc sql;
     create table FlightClusterExport as
          select c.newFlightId, c.Cluster10
               from Work.ClusterFinal as c
                    where c.date_in_range = 1;
/*PROC EXPORT DATA= WORK.FlightClusterExport
            OUTFILE= "C:\Users\Jeff\Analysis\Mesoscopic Flow Model\Sect...
...o
r Traversal Time Variation\Traversal Analysis\flightcluster.csv"
            DBMS=CSV REPLACE;
RUN;*/
```

MATLAB Script to Create Box-and-Whisker Diagrams (error_box_plots.m)

The following MATLAB script is used to create the box-and-whisker diagrams shown in Figures 18 and 20

```
% Create box plots for error at 30 minute look-ahead
err = csvread('errors_cluster_airport.csv');
% Create a cluster label
cluster_label = cell(length(err),1);
airport_label = cell(length(err),1);
ordered_cluster = cell(length(err),1);
for i=1:length(err)
   if (i <= 5)
       if(i == 1)
           airport_label(i) = {'Large Hub '};
       elseif(i == 2)
           airport_label(i) = {'Medium Hub'};
       elseif(i == 3)
           airport_label(i) = {'Small Hub '};
       elseif(i == 4)
           airport_label(i) = {'Non-Hub   '};
       elseif(i == 5)
           airport_label(i) = {'Foreign   '};
       end
   elseif(err(i,3) == 1)
       airport_label(i) = {'Large Hub '};
   elseif(err(i,3) == 2)
       airport_label(i) = {'Medium Hub'};
   elseif(err(i,3) == 3)
       airport_label(i) = {'Small Hub '};
   elseif(err(i,3) == 4)
       airport_label(i) = {'Non-Hub   '};
   elseif(err(i,3) == 5)
       airport_label(i) = {'Foreign   '};
   end

   if(err(i,2) < 10)
       tmp = ' ';
   else
       tmp = '';
   end
   cluster_label(i) = {['Cluster ' tmp num2str(err(i,2))]};

   if(i < 11)
       tmp2 = ' ';
       if(i == 10)
           tmp2 = '';
       end
```

```
        ordered_cluster(i) = {['Cluster ' tmp2 num2str(i)]};
    elseif(err(i,2) == 1)
        ordered_cluster(i) = {'Cluster  8'};
    elseif(err(i,2) == 2)
        ordered_cluster(i) = {'Cluster  3'};
    elseif(err(i,2) == 3)
        ordered_cluster(i) = {'Cluster  4'};
    elseif(err(i,2) == 4)
        ordered_cluster(i) = {'Cluster  7'};
    elseif(err(i,2) == 5)
        ordered_cluster(i) = {'Cluster  9'};
    elseif(err(i,2) == 6)
        ordered_cluster(i) = {'Cluster  5'};
    elseif(err(i,2) == 7)
        ordered_cluster(i) = {'Cluster  6'};
    elseif(err(i,2) == 8)
        ordered_cluster(i) = {'Cluster  1'};
    elseif(err(i,2) == 9)
        ordered_cluster(i) = {'Cluster 10'};
    elseif(err(i,2) == 10)
        ordered_cluster(i) = {'Cluster  2'};
    end
end
boxplot(err(:,1),airport_label)
```

SAS Script to Create Histogram of Departure Time Prediction Errors by Airport Type (create_hists.sas)

The following SAS script is used to create the histograms such as that depicted in Figure 21.

```
proc sql;
     create table deperrors as
          select d._Flight, d._Departing_Hub_Size, d._Arriving_Hub_Size...
...,
               d._Origin_Airport, d._Destination_Airport, d._0minLA,
               d._5minLA, d._15minLA, d._30minLA, d._60minLA, d._120min...
...LA
                    from Work.Departureerrors as d;
/* Create a unique list of large hubs */
proc sql;
     create table largehubs as
          select distinct d._Origin_Airport from
               Work.Deperrors as d
                    where d._Departing_Hub_Size = "Large";
/* Filter data before creating histograms */
proc sql;
     create table Deperrors_ as
          select * from Work.Deperrors as d
               where d._Departing_Hub_Size = "Foreign";
               /*where d._Origin_Airport = "TPA" or d._Origin_Airport =...
... "KTPA";*/

/* Create histogram */
proc univariate data = Work.Deperrors_;
     where _0minLA > -60 and _0minLA < 90;
     var _0minLA;
     histogram / midpoints = -60 to 90 by 1
                    outhist=OutHist0;
run;
proc univariate data = Work.Deperrors_;
     where _5minLA > -60 and _5minLA < 90;
     var _5minLA;
     histogram / midpoints = -60 to 90 by 1
                    outhist=OutHist5;
run;
proc univariate data = Work.Deperrors_;
     where _15minLA > -60 and _15minLA < 90;
     var _15minLA;
     histogram / midpoints = -60 to 90 by 1
                    outhist=OutHist15;
run;
proc univariate data = Work.Deperrors_;
     where _30minLA > -60 and _30minLA < 90;
```

```
     var _30minLA;
     histogram / midpoints = -60 to 90 by 1
                    outhist=OutHist30;
run;
proc univariate data = Work.Deperrors_;
     where _60minLA > -60 and _60minLA < 90;
     var _60minLA;
     histogram / midpoints = -60 to 90 by 1
                    outhist=OutHist60;
run;
proc univariate data = Work.Deperrors_;
     where _120minLA > -60 and _120minLA < 90;
     var _120minLA;
     histogram / midpoints = -60 to 90 by 1
                    outhist=OutHist120;
run;
/* Reformat the output histogram */
proc sql;
     create table OutHist2 as
          select o0._MIDPT_, o0._OBSPCT_ as LAT0min, o5._OBSPCT_ as LAT...
...5min,
          o15._OBSPCT_ as LAT15min, o30._OBSPCT_ as LAT30min, o60._OBSP...
...CT_ as LAT60min,
          o120._OBSPCT_ as LAT120min
          from Work.OutHist0 as o0, Work.OutHist5 as o5, Work.OutHist15...
... as o15,
               Work.OutHist30 as o30, Work.OutHist60 as o60, Work.OutHi...
...st120 as o120
                    where o0._MIDPT_ = o5._MIDPT_ and o0._MIDPT_ = o15....
..._MIDPT_ and
                         o0._MIDPT_ = o30._MIDPT_ and o0._MIDPT_ = o60....
..._MIDPT_ and
                         o0._MIDPT_ = o120._MIDPT_;
/* Save the results */
PROC EXPORT DATA= WORK.Outhist2
            OUTFILE= "C:\Users\Jeff\Analysis\Departure Time Variation\N...
...e
w Fitting\Histogram by Airport and Airport Type\ForeignHist.csv"
            DBMS=CSV REPLACE;
RUN;
```

MATLAB Script to Fit the Kernel Density Estimate for the Departure Time and En Route Histograms (calculate_kernel_densities.m)

The following MATLAB script fits the kernel density estimates for both airport departure prediction and en route traversal time.

```
% Create kernel density estimates for airport departure uncertainty at ...
...% different look ahead times and for en route airspace
tic
fid = fopen('kernel_densities.csv', 'wt');
% Get the operating system
import java.lang.*;
osName = System.getProperty('os.name');
isWin = osName.equals('Windows XP');
isMac = ~isWin;
if isMac
    %javaclasspath('/Users/antoniotrani/Documents/Jeff/Programming/AirP...
...lanJ/build/classes')
    airportDepartureUncertaintyDir = '/Users/antoniotrani/Documents/Jef...
...f/Analysis/Departure Time Variation/New Fitting/Histogram by Airport a...
...nd Airport Type/';
    sectorTraversalUncertaintyDir = '/Users/antoniotrani/Documents/Jeff...
.../Analysis/Mesoscopic Flow Model/Sector Traversal Time Variation/Traver...
...sal Analysis/';
else
    %javaclasspath('C:\\Documents and Settings\\jeff\\My Documents\\Jav...
...a Projects\\AirPlanJ\\build\\classes\\')
    airportDepartureUncertaintyDir = '';
    sectorTraversalUncertaintyDir = '';
end
% Range of density estimate
xi = -90:.5:120;
% Get the names of the files in the airport directory
airportFiles = dir(airportDepartureUncertaintyDir);
% Calculate kernel density
for i=1:length(airportFiles)
   airportFileName = airportFiles(i).name;

   airportFileNameS = java.lang.String(airportFileName);

   if(not(airportFileNameS.endsWith('csv')))
       continue;
   end

   airportFileNameS = airportFileNameS.substring(0,airportFileNameS.len...
...gth-8);

   hist = csvread([airportDepartureUncertaintyDir airportFileName],1,0)...
```

```
...;

   time = hist(:,1);
   lat0 = round(1000*hist(:,2));
   lat5 = round(1000*hist(:,3));
   lat15 = round(1000*hist(:,4));
   lat30 = round(1000*hist(:,5));
   lat60 = round(1000*hist(:,6));
   lat120 = round(1000*hist(:,7));

   lat0_replications = zeros(sum(lat0),1);
   lat5_replications = zeros(sum(lat5),1);
   lat15_replications = zeros(sum(lat15),1);
   lat30_replications = zeros(sum(lat30),1);
   lat60_replications = zeros(sum(lat60),1);
   lat120_replications = zeros(sum(lat120),1);

   k0 = 1;
   k5 = 1;
   k15 = 1;
   k30 = 1;
   k60 = 1;
   k120 = 1;

   for j=1:length(hist)
       for k=1:lat0(j)
          lat0_replications(k0) = time(j);
          k0 = k0 + 1;
       end

       for k=1:lat5(j)
          lat5_replications(k5) = time(j);
          k5 = k5 + 1;
       end

       for k=1:lat15(j)
          lat15_replications(k15) = time(j);
          k15 = k15 + 1;
       end

       for k=1:lat30(j)
          lat30_replications(k30) = time(j);
          k30 = k30 + 1;
       end

       for k=1:lat60(j)
          lat60_replications(k60) = time(j);
```

```
            k60 = k60 + 1;
        end

        for k=1:lat120(j)
            lat120_replications(k120) = time(j);
            k120 = k120 + 1;
        end
    end

    f0 = ksdensity(lat0_replications,xi);
    f5 = ksdensity(lat5_replications,xi);
    f15 = ksdensity(lat15_replications,xi);
    f30 = ksdensity(lat30_replications,xi);
    f60 = ksdensity(lat60_replications,xi);
    f120 = ksdensity(lat120_replications,xi);

    % Normalize the data
    f0 = f0/sum(f0);
    f5 = f5/sum(f5);
    f15 = f15/sum(f15);
    f30 = f30/sum(f30);
    f60 = f60/sum(f60);
    f120 = f120/sum(f120);


    for m=1:length(f0)
        fprintf(fid, '%s, 0, %4f, %10f\n', char(airportFileNameS), xi(m)...
...., f0(m));
    end

    for m=1:length(f5)
        fprintf(fid, '%s, 5, %4f, %10f\n', char(airportFileNameS), xi(m)...
..., f5(m));
    end

    for m=1:length(f15)
        fprintf(fid, '%s, 15, %4f, %10f\n', char(airportFileNameS), xi(m...
...), f15(m));
    end

    for m=1:length(f30)
        fprintf(fid, '%s, 30, %4f, %10f\n', char(airportFileNameS), xi(m...
...), f30(m));
    end

    for m=1:length(f60)
        fprintf(fid, '%s, 60, %4f, %10f\n', char(airportFileNameS), xi(m...
```

```
...), f60(m));
   end

   for m=1:length(f120)
       fprintf(fid, '%s, 120, %4f, %10f\n', char(airportFileNameS), xi(...
...m), f120(m));
   end

   %break;

end
xi2 = -2:.1:5;
for i=1:5
   if(i==1)
       fileName = 'ratiosLT4';
   elseif(i==2)
       fileName = 'ratiosGT4LT8';
   elseif(i==3)
       fileName = 'ratiosGT8LT12';
   elseif(i==4)
       fileName = 'ratiosGT12LT16';
   elseif(i==5)
       fileName = 'ratiosGT16';
   end

   data = csvread([sectorTraversalUncertaintyDir fileName '.csv']);

   f = ksdensity(data(:,3),xi2);

   f = f/sum(f);

   for m=1:length(f)
      fprintf(fid,'%s, 0, %4f, %10f\n', fileName, xi2(m), f(m));
   end

end
fclose(fid);
toc
```

MATLAB Script for Demand Calculations to Compare Deterministic and Stochastic Demand Errors (traversal_analysis.m)

The following MATLAB script implements the methods of Section 5.4 to compare the deterministic and probabilistic sector demand estimates.

```
% traversal_analysis.m
function traversal_analysis
tic
clear java
clear all
% Get the operating system
import java.lang.*;
import java.util.*;
import java.io.*;
osName = System.getProperty('os.name');
isWin = osName.equals('Windows XP');
isMac = ~isWin;
if isMac
    javaclasspath('/Users/antoniotrani/Documents/Jeff/Programming/AirPl...
...anJ/build/classes')
    plannedCrossingFileName = '/Users/antoniotrani/Documents/Jeff/Analy...
...sis/Mesoscopic Flow Model/Sector Traversal Time Variation/Traversal An...
...alysis/sectorcrossing_flightplan.out.1';
    observedCrossingFileName = '/Users/antoniotrani/Documents/Jeff/Anal...
...ysis/Mesoscopic Flow Model/Sector Traversal Time Variation/Traversal A...
...nalysis/sectorcrossing_tzradar.out.1';
    sectorAreaVolumeFileName = '/Users/antoniotrani/Documents/Jeff/Prog...
...ramming/AirPlanJ/data/sector/sector_area.csv';
    airportTypeFileName = '/Users/antoniotrani/Documents/Jeff/Programmi...
...ng/AirPlanJ/data/airports/airportinfo.csv';
    kernelDensityFileName = '/Users/antoniotrani/Documents/Jeff/Program...
...ming/AirPlanJ/data/kernel/kernel_densities.csv';
    ramsFlightPlanFile = '/Users/antoniotrani/Documents/Jeff/Analysis/M...
...esoscopic Flow Model/Sector Traversal Time Variation/050829 RAMS Fligh...
...t Plan/traffic.dat';
else
    javaclasspath('C:\\Documents and Settings\\jeff\\My Documents\\Java...
... Projects\\AirPlanJ\\build\\classes\\')
    plannedCrossingFileName = 'C:\Documents and Settings\Jeff\My Docume...
...nts\Drop Box\Analysis\Mesoscopic Flow Model\Sector Traversal Time Vari...
...ation\Traversal Analysis\sectorcrossing_flightplan.out.1';
    observedCrossingFileName = 'C:\Documents and Settings\Jeff\My Docum...
...ents\Drop Box\Analysis\Mesoscopic Flow Model\Sector Traversal Time Var...
...iation\Traversal Analysis\sectorcrossing_tzradar.out.1';
    sectorAreaVolumeFileName = 'C:\Documents and Settings\Jeff\My Docum...
...ents\Java Projects\AirPlanJ\data\sector\sector_area.csv';
    airportTypeFileName = 'C:\Documents and Settings\Jeff\My Documents\...
```

```
...Java Projects\AirPlanJ\data\airports\airportinfo.csv';
    kernelDensityFileName = 'C:\Documents and Settings\Jeff\My Document...
...s\Java Projects\AirPlanJ\data\kernel\kernel_densities.csv';
    ramsFlightPlanFile = 'C:\Documents and Settings\Jeff\My Documents\D...
...rop Box\Analysis\Mesoscopic Flow Model\Sector Traversal Time Variation...
...\050829 RAMS Flight Plan\traffic.dat';
end
outFile = [pwd filesep 'ratios.csv'];
import FlightTraversal.*
import SectorTraversal.*
import TraversalReader.*
traversalFlights = TraversalReader.createTraversalFlights;
traversalSectors = TraversalReader.createTraversalSectors;
traversalReader = TraversalReader(traversalFlights, traversalSectors);
traversalReader.readPlannedSectorCrossing(plannedCrossingFileName);
traversalReader.readObservedSectorCrossing(observedCrossingFileName);
traversalReader.readSectorAreaVolume(sectorAreaVolumeFileName);
traversalWriter = TraversalWriter;
traversalWriter.writeTraversalRatioAndDensity(outFile, traversalFlights...
..., traversalSectors);
kernelReader = KernelReader;
kernelDensities = kernelReader.readKernelDensities(kernelDensityFileNam...
...e);
% Read airport type
airportTypes = traversalReader.readAirportTypes(airportTypeFileName);
% Read airport from flight plan
traversalReader.readAirports(ramsFlightPlanFile);
% Deterministic sector demand calculations for 10, 30, and 60 min
% look-ahead times
% Pseudo-code for 10 min LAT
% For each flight in traversalFlights
% For each sector in flight.plannedSectors
%   If first sector in list then
%       Use planned demand
%   Else if second or later sector in the planned list then
%       Find a previous sector with the closest match in observed and
%       planned lists
%           If planned previous entry - current entry is less than 10
%           minutes than don't consider
%           Else:
%               - prevObsEntry == observed entry to previous sector
%               - prevPlanEntry == planned entry to previous sector
%               - currPlanEntry == planned entry to current sector
%               - currCorrEntry == corrected entry to current sector
%                   currCorrEntry = currPlanEntry +
%                   (prevObsEntry-prevPlanEntry)
%               - Add demand in sector from (currCorrEntry) to
```

```
%                 (currCorrEntry + plannedTraversalTime)
numberFlights = traversalFlights.values.size;
numberIterations = numberFlights*3;
counter = 0;
for l=1:3
    % Set look-ahead time (LAT)
    if(l == 1)
        LAT = 10;
    elseif(l == 2)
        LAT = 30;
    else %(l == 3)
        LAT = 60;
    end
    e = traversalFlights.values.iterator;
    while(e.hasNext)
        flight = e.next;
        plannedSectors = flight.PlannedSector;
        observedSectors = flight.ObservedSector;
        % Iterate through planned sectors
        for i=0:plannedSectors.size - 1
            previousSectorFound = false;
            currentPlannedEntry = flight.PlannedEntryMin.get(i);
            % Search for a previous sector more than 10 minutes away fr...
...om
            % current sector
            for j=0:i
                j2 = i-j;
                candidatePreviousSector = plannedSectors.get(j2);
                previousPlannedEntry = flight.PlannedEntryMin.get(j2);
                if(currentPlannedEntry - previousPlannedEntry < LAT)
                    continue;
                end
                % Find the observed sector with the closest entry time
                diffTime = 99999;
                for k=0:observedSectors.size - 1
                    obsSector = observedSectors.get(k);
                    if(not(strcmp(obsSector,candidatePreviousSector)))
                        continue;
                    end
                    previousObservedEntry = flight.ObservedEntryMin.get...
...(k);
                    if(abs(previousPlannedEntry - previousObservedEntry...
...) < diffTime)
                        diffTime = abs(previousPlannedEntry - previousO...
...bservedEntry);
                        selectedIndex = k;
                    end
```

```
                end
                if(diffTime < 99999)
                    previousSectorFound = true;
                    previousSectorPlannedIdx = j2;
                    break;
                end
            end
            % If information greater than 10 minutes previous found use...
... the
            % updated information
            if(previousSectorFound)
                currentCorrectedEntry = currentPlannedEntry + (previous...
...ObservedEntry -...
                    previousPlannedEntry);

                k = previousSectorPlannedIdx;

                % For k to i (-1?)
                if(k == i-1)
                    traversalTime = flight.PlannedTraversalTimeMin.get(...
...k);
                    errorDistribution = get_error_distribution_from_tra...
...versaltime(...
                        traversalTime, kernelDensities, false);
                else
                    for i2 = k+1:i-1
                        if(i2 == k+1)
                            traversalTime = flight.PlannedTraversalTime...
...Min.get(k);
                            errorDistribution = get_error_distribution_...
...from_traversaltime(...
                                traversalTime, kernelDensities, false);...
...                            end
                        traversalTime1 = flight.PlannedTraversalTimeMin...
....get(i2);
                        errorDistribution1 = get_error_distribution_fro...
...m_traversaltime(...
                            traversalTime1, kernelDensities, false);

                        errorDistribution = distribution_convolution(.....
....
                            errorDistribution, errorDistribution1);
                    end
                end

            else % Used the original planned information
                currentCorrectedEntry = currentPlannedEntry;
```

```
                % Get the type for this airport
                airportType = airportTypes.get(flight.Airport);

                % Use foreign as a default
                if(isempty(airportType))
                   airportType = 'Foreign';
                end

                errorDistribution1 = kernelDensities.get(airportType);

                errorDistribution = cell2mat(cell(errorDistribution1.de...
...nsityArray));
            end
            plannedTraversal = flight.PlannedTraversalTimeMin.get(i);
            currentCorrectedExit = currentCorrectedEntry + plannedTrave...
...rsal;
            currentCorrectedEntry = floor(currentCorrectedEntry);
            currentCorrectedExit = floor(currentCorrectedExit);

            % Convolute error distribution and
            errorDist2 = get_error_distribution_from_traversaltime(...
                plannedTraversal, kernelDensities, true);
            errorDistribution = distribution_convolution(errorDistribut...
...ion,...
                errorDist2);
            sector = traversalSectors.get(plannedSectors.get(i));
            if(isempty(sector))
                continue;
            end
            % Do deterministic demand calculations
            sector.addDeterministicDemand(LAT, ...
                currentCorrectedEntry, currentCorrectedExit);

            % Do stochastic demand calculations
            xc = -90:0.5:120;
            sector.addProbabilisticDemand(LAT, currentCorrectedEntry, x...
...c,...
                errorDistribution, flight.FlightId);
        end

        counter = counter + 1;

        if(mod(counter,100) == 0)
           disp(['Iteration: ' num2str(counter) ' of ' num2str(numberIt...
...erations)]);
        end
```

```
    end
end
% Write results for analysis in SAS
errorFile = [pwd filesep 'demand_error.csv'];
traversalWriter.writeTraversalError(errorFile, traversalFlights, traver...
...salSectors);
toc
```

MATLAB Supporting Script for Demand Calculations (distribution_convolution.m)

The script distribution_convolution.m is called from traversal_analysis.m.

```
function outDist = distribution_convolution(inDist1, inDist2);
% inDistX: -90:0.5:120 range
outDist_tmp = conv(inDist1, inDist2);
% outDist_tmp: -180:0.5:240
outDist = outDist_tmp(181:601);
% outDist: -90:0.5:120
```

MATLAB Supporting Script for Demand Calculations (get_error_distribution _from_traversaltime.m)

The script get_error_distribution_from_traversaltime.m is called from traversal_analysis.m.

```
function err = get_error_distribution_from_traversaltime(traversalTime,...
... ...
    kernelDensities, returnTraversalDistribution)
if(traversalTime < 0.5)
   err = zeros(length(-90:0.5:120),1);
   err(181) = 1;
   return;
end
if(traversalTime < 4)
    kernelDensity = kernelDensities.get('ratiosLT4');
elseif(traversalTime < 8)
    kernelDensity = kernelDensities.get('ratiosGT4LT8');
elseif(traversalTime < 12)
    kernelDensity = kernelDensities.get('ratiosGT8LT12');
elseif(traversalTime < 16)
    kernelDensity = kernelDensities.get('ratiosGT12LT16');
else
    kernelDensity = kernelDensities.get('ratiosGT16');
end
kernelDensity2 = zeros(kernelDensity.noPoints,1);
for i=1:kernelDensity.noPoints
   kernelDensity2(i) = kernelDensity.getDensity(i-1);
end
xi = -2:.1:5;
% xi*traversalTime - traversalTime
if(returnTraversalDistribution)
    xi = xi*traversalTime;
else
    xi = (xi-1)*traversalTime;
end
% Output x values
xi2 = -90:0.5:120;
err = interp1(xi,kernelDensity2,xi2,'linear',0);
% Normalize the distribution
if(returnTraversalDistribution)
    err = err*traversalTime/sum(err);
else
    err = err/sum(err);
end
```

APPENDIX C: CAPACITY ANALYSIS SCRIPTS AND DATA

This appendix contains the scripts used in the capacity analysis. The following lists the files included in this appendix.

MATLAB Script for Real Time Verification System (RTVS) Image Processing (get_convection_rtvs.m)

The following MATLAB script downloads the RTVS convection forecast image and processes the image to obtain the forecast, observed convection, 10 nm buffer, and 20 nm buffer around the observed convection.

```
% Get locations of convection from RTVS image
% Website: http://rtvs.noaa.gov/conv/
%
% Recorded variables:
% - num_ncwf_cells        : Forecast cells
% - num_conv_buffer_20nm  : Buffer cells to 20nm outside convection
% - num_conv_buffer_10nm  : Buffer cells to 10nm outside convection
% - num_conv_cells        : Convection cells
% - centroid_dist_nm      : Distance between forecast convection locati...
...on
%                           and observed convection location centroid.
tic
clear;
% Add path to the image directory
images_dir = [pwd filesep 'images'];
addpath(images_dir);
file_list = dir(fullfile(images_dir,'*.gif'));
num_images = length(file_list);
% Number of images
%num_convection_images = 1;
% Days in month (2006)
days_in_month = [31 28 31 30 31 30 31 31 30 31 30 31];
% Date range
year = 2006;
start_month = 6;
end_month = 7;
%for img=1:num_images
for month_i = start_month:end_month
    if month_i < 10
        month_string = ['0' num2str(month_i)];
    else
        month_string = num2str(month_i);
    end

    for day_i = 1:days_in_month(month_i)
        if day_i < 10
            day_string = ['0' num2str(day_i)];
        else
            day_string = num2str(day_i);
        end
```

```
        saved_file_name = [pwd filesep 'results' filesep 'detailcellcou...
...nt2006' month_string day_string '.mat'];
        if(exist(saved_file_name,'file'))
           continue;
        end
        % Counter
        img = 0;
        img_num_ncwf_cells = zeros(24,1);
        img_num_conv_buffer_20nm = zeros(24,1);
        img_num_conv_buffer_10nm = zeros(24,1);
        img_num_conv_cells = zeros(24,1);
        img_centroid_dist_nm = [];
        ncwf_size = [];
        conv_size = [];
        buffer10nm_size = [];
        buffer20nm_size = [];

        for hour_i = 0:23
            img = img + 1;
            % First check that file does not already exist
            img_file_exists = false;

            if hour_i < 10
                hour_string = ['0' num2str(hour_i) '00'];
            else
                hour_string = [num2str(hour_i) '00'];
            end
            ncwf_filename = ['ncwfp_ncwdp_national_' num2str(year) ...
                month_string day_string '_' hour_string '_1hr_cutoff1.g...
...if'];
            disp(ncwf_filename)
            for i=1:num_images
               if(strcmp(ncwf_filename,file_list(i).name))
                  img_file_exists = true;
                  break;
               end
            end
            %ncwf_filename = 'ncwfp_ncwdp_national_20050829_0000_1hr_cu...
...toff1.gif';
            %ncwf_filename = file_list(img).name;
            % If file does not exists download from RTVS server and sav...
...e
            if(~img_file_exists)
                rtvs_url = ['http://rtvs.noaa.gov/conv/2001/display/'.....
....
                    'scripts/index.cgi?product=ncwfp&ob=ncwdp&region='....
.....
```

```
                    'national&grid_size=4km&year=' num2str(year)...
                    '&month=' month_string '&day=' day_string '&'...
                    'run_time=' hour_string '&forecast_length=1hr&windo...
...w='];
                rtvs_img_url = ['http://rtvs.noaa.gov/conv/2001/display...
.../'...
                    'tempdir/' ncwf_filename];
                s=urlread(rtvs_url);
                [cdata, colourmap_2]=imread(rtvs_img_url,'gif');
                pause(3);
                % Save the image
                imwrite(cdata, colourmap_2,fullfile(images_dir,ncwf_fil...
...ename),'gif');
            else
                % Image Data: cdata, colourmap_2
                [cdata,colourmap_2] = imread(ncwf_filename,'gif');
            end
            [num_rows, num_cols] = size(cdata);
            %
            % To recover the image:
            % >> image(cdata); colormap(colourmap_2); axis image;
            %
            % Colour Values
            % 0: White
            % 1: Black (Outline, States, Lettering)
            % 2: Green (Convection)
            % 4: Yellow (NCWF Forecast)
            % 6: Red (Lettering)
            % 8: Light Brown (Land)
            % 11: Blue (Water)
            % 12: Grey (ARTCC Boundaries)
            %
            % Points to match image locations to geographic latitude/lo...
...ngitude
            % lat_map = [48.38; 45.0014; 41.9938; 40.998; 36.999; 31.33...
...24; 25.11];
            % lat_img = [32; 113; 181; 203; 288; 401; 519];
            % lon_map = [-124.725; -111.0544; -114.041; -111.046; -109....
...044; -109.0494; -81.1];
            % lon_img = [201;385;345;385;412;412;789];
            BLACK_COLOR_IDX = 1;
            CONVECTION_COLOR_IDX = 2;
            NCWF_COLOR_IDX = 4;
            BUFFER_COLOR_IDX = 5;
            LAND_COLOR_IDX = 8;
            ARTCC_COLOR_IDX = 12;
            MIN_IMAGE_X = 160;  % Exclude the legend
```

```
            % Copy of original image
            cdata_orig = cdata;
            % NCWF cells processed for convection
            cdata_conv = cdata;
            for k=1:3
               for i=1+5:550-5
                  for j=MIN_IMAGE_X+5:1000-5
                      % Check for ARTCC boundaries
                      if(k == 1 && cdata_conv(i,j) == ARTCC_COLOR_IDX)
                         if(cdata_conv(i+1,j) == NCWF_COLOR_IDX || ...
                                 cdata_conv(i-1,j) == NCWF_COLOR_IDX ||...
... ...
                                 cdata_conv(i,j+1) == NCWF_COLOR_IDX ||...
... ...
                                 cdata_conv(i,j-1) == NCWF_COLOR_IDX)
                             if(cdata_conv(i+1,j) == LAND_COLOR_IDX || ...
......
                                 cdata_conv(i-1,j) == LAND_COLOR_IDX ||...
... ...
                                 cdata_conv(i,j+1) == LAND_COLOR_IDX ||...
... ...
                                 cdata_conv(i,j-1) == LAND_COLOR_IDX)
                                cdata_conv(i,j) = LAND_COLOR_IDX;
                             elseif(cdata_conv(i+1,j) == BLACK_COLOR_ID...
...X || ...
                                 cdata_conv(i-1,j) == BLACK_COLOR_IDX |...
...| ...
                                 cdata_conv(i,j+1) == BLACK_COLOR_IDX |...
...| ...
                                 cdata_conv(i,j-1) == BLACK_COLOR_IDX)
                                cdata_conv(i,j) = BLACK_COLOR_IDX;
                             else
                                cdata_conv(i,j) = NCWF_COLOR_IDX;
                             end
                         end
                      end
                      % Check for black outlines
                      if(cdata_conv(i,j) == BLACK_COLOR_IDX)
                         % Single black lines
                         if(cdata_conv(i+1,j) == NCWF_COLOR_IDX && ...
                                 cdata_conv(i-1,j) == NCWF_COLOR_IDX)
                            cdata_conv(i,j) = NCWF_COLOR_IDX;
                         end
                         if(cdata_conv(i,j+1) == NCWF_COLOR_IDX && ...
                                 cdata_conv(i,j-1) == NCWF_COLOR_IDX)
                            cdata_conv(i,j) = NCWF_COLOR_IDX;
                         end
```

```
                        % Double black lines
                        if(cdata_conv(i+2,j) == NCWF_COLOR_IDX && ...
                                cdata_conv(i+1,j) == BLACK_COLOR_IDX &...
...& ...
                                cdata_conv(i-1,j) == NCWF_COLOR_IDX)
                           cdata_conv(i,j) = NCWF_COLOR_IDX;
                           cdata_conv(i+1,j) = NCWF_COLOR_IDX;
                        end
                        if(cdata_conv(i,j+2) == NCWF_COLOR_IDX && ...
                                cdata_conv(i,j+1) == BLACK_COLOR_IDX &...
...& ...
                                cdata_conv(i,j-1) == NCWF_COLOR_IDX)
                           cdata_conv(i,j) = NCWF_COLOR_IDX;
                           cdata_conv(i,j+1) = NCWF_COLOR_IDX;
                        end
                     end
                     % Change convection to NCWF if adjacent
                     if(cdata_conv(i,j) == CONVECTION_COLOR_IDX)
                        if(cdata_conv(i+1,j) == NCWF_COLOR_IDX || ...
                                cdata_conv(i-1,j) == NCWF_COLOR_IDX ||...
... ...
                                cdata_conv(i,j+1) == NCWF_COLOR_IDX ||...
... ...
                                cdata_conv(i,j-1) == NCWF_COLOR_IDX)
                           cdata_conv(i,j) = NCWF_COLOR_IDX;
                        end
                     end
                  end
               end
            end
            % Display the image processed for convection
            %colormap(colourmap_2)
            %image(cdata_conv)
            % Group the NCWF cells
            cdata_ncwf = zeros(num_rows, num_cols);
            convection_group_num = 0;
            for k=1:50
               for i=1:num_rows
                  for j=MIN_IMAGE_X:num_cols
                     if(cdata_conv(i,j) == NCWF_COLOR_IDX)
                        if(cdata_ncwf(i+1,j) > 0)
                           cdata_ncwf(i,j) = cdata_ncwf(i+1,j);
                        elseif(cdata_ncwf(i-1,j) > 0)
                           cdata_ncwf(i,j) = cdata_ncwf(i-1,j);
                        elseif(cdata_ncwf(i,j+1) > 0)
                           cdata_ncwf(i,j) = cdata_ncwf(i,j+1);
                        elseif(cdata_ncwf(i,j-1) > 0)
```

```
                        cdata_ncwf(i,j) = cdata_ncwf(i,j-1);
                    else
                        convection_group_num = convection_group_num...
... + 1;
                        cdata_ncwf(i,j) = convection_group_num;
                    end
                    if(cdata_ncwf(i,j) > cdata_ncwf(i+1,j) && ...
                            cdata_ncwf(i+1,j) > 0)
                        cdata_ncwf(i,j) = cdata_ncwf(i+1,j);
                    elseif(cdata_ncwf(i,j) > cdata_ncwf(i-1,j) && ...
......
                            cdata_ncwf(i-1,j) > 0)
                        cdata_ncwf(i,j) = cdata_ncwf(i-1,j);
                    elseif(cdata_ncwf(i,j) > cdata_ncwf(i,j+1) && ...
......
                            cdata_ncwf(i,j+1) > 0)
                        cdata_ncwf(i,j) = cdata_ncwf(i,j+1);
                    elseif(cdata_ncwf(i,j) > cdata_ncwf(i,j-1) && ...
......
                            cdata_ncwf(i,j-1) > 0)
                        cdata_ncwf(i,j) = cdata_ncwf(i,j-1);
                    end
                end
            end
        end
    end
    cdata_ncwf2 = cdata_ncwf;
    for i=1:num_rows
        for j=MIN_IMAGE_X:num_cols
            % Use blue for even
            if(cdata_ncwf(i,j) > 0 && mod(cdata_ncwf(i,j),2) ==...
... 0)
                cdata_ncwf2(i,j) = 11;
            % Use red for odd
            elseif(cdata_ncwf(i,j) > 0)
                cdata_ncwf2(i,j) = 6;
            end
        end
    end
    %image(cdata_ncwf2)
    % Number of convection image cells
    num_conv_cells = 0;
    for i=1:550
        for j=MIN_IMAGE_X:1000
            if(cdata_orig(i,j) == CONVECTION_COLOR_IDX)
               num_conv_cells = num_conv_cells + 1;
            end
```

```
                end
            end
            %
            % convection_lat = zeros(num_conv_cells,1);
            % convection_lon = zeros(num_conv_cells,1);
            % Transform from image coordinates to latitude/longitude
            % Coefficients from curve fitting lat_map & lat_img, lon_ma...
...p & lon_img
            % k=1;
            % for i=1:550
            %     for j=MIN_IMAGE_X:1000
            %         if cdata(i,j) == CONVECTION_COLOR_IDX
            %             convection_lat(k) = (1.5973e-11)*i^4-(1.2599e...
...-8)*i^3- ...
            %                 (1.2157e-5)*i^2-0.039851*i+49.669;
            %             convection_lon(k) = (-3.7111e-9)*j^3+(4.8618e...
...-6)*j^2+...
            %                 0.072428*j-139.45;
            %             k=k+1;
            %         end
            %     end
            % end
            cdata_trans_lat = zeros(num_rows, num_cols);
            cdata_trans_lon = zeros(num_rows, num_cols);
            for i=1:num_rows
                for j=MIN_IMAGE_X:num_cols
                    cdata_trans_lat(i,j) = (1.5973e-11)*i^4-(1.2599e-8)...
...*i^3- ...
                       (1.2157e-5)*i^2-0.039851*i+49.669;
                    cdata_trans_lon(i,j) = (-3.7111e-9)*j^3+(4.8618e-6)...
...*j^2+...
                       0.072428*j-139.45;
                end
            end
            % Get the average lat/lon for each convection number (NCWF ...
...forecast area)
            % latitude; longitude; count
            avg_ncwf = zeros(convection_group_num,3);
            for i=1:num_rows
                for j=MIN_IMAGE_X:num_cols
                    idx = cdata_ncwf(i,j);
                    if(idx > 0)
                       avg_ncwf(idx,1) = avg_ncwf(idx,1) + cdata_trans_...
...lat(i,j);
                       avg_ncwf(idx,2) = avg_ncwf(idx,2) + cdata_trans_...
...lon(i,j);
                       avg_ncwf(idx,3) = avg_ncwf(idx,3) + 1;
```

```
                end
            end
        end
        ln_avg = size(avg_ncwf,1);
        for i=1:ln_avg
           j = ln_avg - i + 1;
           if(avg_ncwf(j,3) == 0)
              avg_ncwf(j,:) = [];
           end
        end
        avg_ncwf(:,1) = avg_ncwf(:,1)./avg_ncwf(:,3);
        avg_ncwf(:,2) = avg_ncwf(:,2)./avg_ncwf(:,3);
        % Count of NCWF forecast cells
        num_ncwf_cells = sum(avg_ncwf(:,3));
        % For each convection cell find the nearest NCWF forecast a...
...nd record the
        % relative angle, distance, forecast index, count, latitude...
..., and longitude.
        conv_stats = zeros(num_conv_cells,6);
        k=1;

        if(size(avg_ncwf,1) < 5)
            % Too little convection.
            img_num_ncwf_cells(img) = 1;
            img_num_conv_buffer_20nm(img) = 1;
            img_num_conv_buffer_10nm(img) = 1;
            img_num_conv_cells(img) = 1;
            continue;
        end

        TRI = delaunay(avg_ncwf(:,1), avg_ncwf(:,2));
        for i=1:num_rows
            for j=MIN_IMAGE_X:num_cols
                if(cdata_orig(i,j) == CONVECTION_COLOR_IDX)
                    K = dsearch(avg_ncwf(:,1),avg_ncwf(:,2),TRI,......
...                         cdata_trans_lat(i,j),cdata_trans_lon(i,j));...
...                     lat1 = cdata_trans_lat(i,j);
                    lon1 = cdata_trans_lon(i,j);
                    lat2 = avg_ncwf(K,1);
                    lon2 = avg_ncwf(K,2);
                    relative_angle = azimuth(lat1, lon1, lat2, lon2...
...);
                    dist = distance(lat1, lon1, lat2, lon2);
                    dist_nm =deg2nm(dist);
                    count = avg_ncwf(K,3);
                    conv_stats(k,1) = relative_angle;
                    conv_stats(k,2) = dist_nm;
```

```
                    conv_stats(k,3) = K;
                    conv_stats(k,4) = count;
                    conv_stats(k,5) = lat1;
                    conv_stats(k,6) = lon1;
                    k=k+1;
                end
            end
        end
        % Update number of convection cells
        num_conv_cells = size(conv_stats,1);
        if (size(conv_stats,1) < 5)
            % Not enough observed convection.
            img_num_ncwf_cells(img) = 1;
            img_num_conv_buffer_20nm(img) = 1;
            img_num_conv_buffer_10nm(img) = 1;
            img_num_conv_cells(img) = 1;
            continue;
        end

        % Calculate the centroid of each convection group
        avg_conv = zeros(size(avg_ncwf,1),3);
        for i=1:num_conv_cells
            idx = conv_stats(i,3);
            lat = conv_stats(i,5);
            lon = conv_stats(i,6);
            avg_conv(idx,1) = avg_conv(idx,1) + lat;
            avg_conv(idx,2) = avg_conv(idx,2) + lon;
            avg_conv(idx,3) = avg_conv(idx,3) + 1;
        end
        avg_conv(:,3) = max(avg_conv(:,3),1);
        avg_conv(:,1) = avg_conv(:,1)./avg_conv(:,3);
        avg_conv(:,2) = avg_conv(:,2)./avg_conv(:,3);
        % Calculate the distance between the forecast centroid and ...
...the convection
        % centroid
        centroid_dist = distance(avg_ncwf(:,1),avg_ncwf(:,2),avg_co...
...nv(:,1),...
            avg_conv(:,2));
        centroid_dist_nm = deg2nm(centroid_dist);
        % Find the minimum and maximum extent of convection to redu...
...ct computations
        min_conv_x = num_cols;
        max_conv_x = MIN_IMAGE_X;
        min_conv_y = num_rows;
        max_conv_y = 0;
        for i=1:num_rows
            for j=MIN_IMAGE_X:num_cols
```

```
                if(cdata_orig(i,j) == CONVECTION_COLOR_IDX)
                   if(j < min_conv_x)
                       min_conv_x = j;
                   end
                   if(j > max_conv_x)
                       max_conv_x = j;
                   end
                   if(i < min_conv_y)
                       min_conv_y = i;
                   end
                   if(i > max_conv_y)
                       max_conv_y = i;
                   end
                end
            end
        end
        min_conv_x = min_conv_x - 5;
        max_conv_x = max_conv_x + 5;
        min_conv_y = min_conv_y - 5;
        max_conv_y = max_conv_y + 5;
        % Add cells within 20nm of convection
        cdata_20nm = cdata;
        cdata_10nm = cdata;
        TRI = delaunay(conv_stats(:,5), conv_stats(:,6));
        K = dsearch(conv_stats(:,5),conv_stats(:,6),TRI,...
            cdata_trans_lat,cdata_trans_lon);
        num_conv_buffer_20nm = 0; % Number of cells within 20nm buf...
...fer
        num_conv_buffer_10nm = 0; % Number of cells within 10nm buf...
...fer
        buffer10nm_cnt = zeros(size(avg_ncwf,1),1);
        buffer20nm_cnt = zeros(size(avg_ncwf,1),1);
        for i=1:num_rows
            %i
            for j=MIN_IMAGE_X:num_cols
                if(cdata_20nm(i,j) ~= CONVECTION_COLOR_IDX && ...
                        j > min_conv_x && ...
                        j < max_conv_x && i > min_conv_y && i < max...
..._conv_y)
                    %K = dsearch(conv_stats(:,5),conv_stats(:,6),TR...
...I,...
                    %cdata_trans_lat(i,j),cdata_trans_lon(i,j));
                    lat1 = cdata_trans_lat(i,j);
                    lon1 = cdata_trans_lon(i,j);
                    lat2 = conv_stats(K(i,j),5);
                    lon2 = conv_stats(K(i,j),6);
                    idx = conv_stats(K(i,j),3);
```

```
                        dist_buffer = distance(lat1, lon1, lat2, lon2);...
...                        dist_buffer_nm = deg2nm(dist_buffer);
                        if(dist_buffer_nm < 20)
                            num_conv_buffer_20nm = num_conv_buffer_20nm...
... + 1;
                            cdata_20nm(i,j) = BUFFER_COLOR_IDX;
                            buffer20nm_cnt(idx) = buffer20nm_cnt(idx) +...
... 1;
                        end
                        if(dist_buffer_nm < 10)
                            num_conv_buffer_10nm = num_conv_buffer_10nm...
... + 1;
                            cdata_10nm(i,j) = BUFFER_COLOR_IDX;
                            buffer10nm_cnt(idx) = buffer10nm_cnt(idx) +...
... 1;
                        end
                    end
                end
            end
            %colormap(colourmap_2)
            %image(cdata_10nm)
            % Record the results
            img_num_ncwf_cells(img) = num_ncwf_cells;
            img_num_conv_buffer_20nm(img) = num_conv_buffer_20nm;
            img_num_conv_buffer_10nm(img) = num_conv_buffer_10nm;
            img_num_conv_cells(img) = num_conv_cells;
            img_centroid_dist_nm = [img_centroid_dist_nm; centroid_dist...
..._nm];
            ncwf_size = [ncwf_size; avg_ncwf(:,3)];
            conv_size = [conv_size; avg_conv(:,3)];
            buffer10nm_size = [buffer10nm_size; buffer10nm_cnt];
            buffer20nm_size = [buffer20nm_size; buffer20nm_cnt];
        end
        bias = img_num_ncwf_cells./img_num_conv_cells;
        bias20nm = img_num_ncwf_cells./(img_num_conv_cells + ...
            img_num_conv_buffer_20nm);
        bias10nm = img_num_ncwf_cells./(img_num_conv_cells + ...
            img_num_conv_buffer_10nm);

        % Remove bias entries of exactly 1 or 0.5 (occurs when no conve...
...ction)
        ln_bias = length(bias);
        for i=1:ln_bias
           j = ln_bias - i + 1;
           if(bias(j) == 1)
               bias(j) = [];
           end
```

```
            if(bias20nm(j) == 0.5)
                bias20nm(j) = [];
            end
            if(bias10nm(j) == 0.5)
                bias10nm(j) = [];
            end
        end

        % Remove excessively large centroid distances
%         ln_centroid = length(img_centroid_dist_nm);
%         for i=1:ln_centroid
%            j = ln_centroid - i + 1;
%            if(img_centroid_dist_nm(j) > 1500)
%               img_centroid_dist_nm(j) = [];
%            end
%         end

        save([pwd filesep 'results' filesep 'bias2006' month_string day...
..._string '.mat'], 'bias', 'bias20nm', 'bias10nm');
        save([pwd filesep 'results' filesep 'cellcount2006' month_strin...
...g day_string '.mat'], 'img_num_ncwf_cells', 'img_num_conv_buffer_20nm'...
...,...
            'img_num_conv_buffer_10nm', 'img_num_conv_cells');
        save([pwd filesep 'results' filesep 'detailcellcount2006' month...
..._string day_string '.mat'], 'ncwf_size', 'conv_size',...
            'buffer10nm_size', 'buffer20nm_size');
        save([pwd filesep 'results' filesep 'centroiddistance2006' mont...
...h_string day_string '.mat'], 'img_centroid_dist_nm');
        clear bias bias20nm bias10nm img_num_ncwf_cells img_num_conv_bu...
...ffer_20nm;
        clear img_num_conv_buffer_10nm img_num_conv_cells img_centroid_...
...dist_nm;
        clear ncwf_size conv_size buffer10nm_size buffer20nm_size;
    end
end
% Combine results
nbias = [];
nbias20nm = [];
nbias10nm = [];
nimg_centroid_dist_nm = [];
nncwf_size = [];
nconv_size = [];
nbuffer10nm_size = [];
nbuffer20nm_size = [];
for month_i = start_month:end_month
    if month_i < 10
        month_string = ['0' num2str(month_i)];
```

```
    else
        month_string = num2str(month_i);
    end

    for day_i = 1:days_in_month(month_i)
        if day_i < 10
            day_string = ['0' num2str(day_i)];
        else
            day_string = num2str(day_i);
        end
        load([pwd filesep 'results' filesep 'bias2006' month_string day...
...._string '.mat']);
        load([pwd filesep 'results' filesep 'cellcount2006' month_strin...
...g day_string '.mat']);
        load([pwd filesep 'results' filesep 'detailcellcount2006' month...
...._string day_string '.mat']);
        load([pwd filesep 'results' filesep 'centroiddistance2006' mont...
...h_string day_string '.mat']);

        nbias = [nbias; bias];
        nbias20nm = [nbias20nm; bias20nm];
        nbias10nm = [nbias10nm; bias10nm];
        nimg_centroid_dist_nm = [nimg_centroid_dist_nm; img_centroid_di...
...st_nm];
        nncwf_size = [nncwf_size; ncwf_size];
        nconv_size = [nconv_size; conv_size];
        nbuffer10nm_size = [nbuffer10nm_size; buffer10nm_size];
        nbuffer20nm_size = [nbuffer20nm_size; buffer20nm_size];
    end
end
% Save the result statistics
save('result_stats.mat','nbias','nbias10nm','nbias20nm','nimg_centroid_...
...dist_nm',...
    'nncwf_size','nconv_size','nbuffer10nm_size','nbuffer20nm_size');
toc
```

MATLAB Main Script that Processes ETMS Radar Track Data to Obtain Aircraft Separations (separation_main.m)

This is the main MATLAB script that calls other functions in the calculation of aircraft separations

```
% Control file for separation calculations
tic
% Load files
isMac = false;
% Specify altitude
flightLevel = 240;
if isMac
    flightsDirectory = ['/Users/antoniotrani/Documents/Jeff/Mesoscopic ...
...Flow Model/Computational Results/Extracted Flights By Flightlevel/FL' ...
...num2str(flightLevel) ' 050829/'];
    windDirectory = '/Users/antoniotrani/Documents/Jeff/AirPlanJ/data/w...
...ind/';
    addpath('/Users/antoniotrani/Documents/Jeff/Mesoscopic Flow Model/M...
...ATLAB Separation/');
    addpath('/Users/antoniotrani/Documents/Jeff/Mesoscopic Flow Model/M...
...ATLAB Java Separation Simulation/');
    javaclasspath('/Users/antoniotrani/Documents/Jeff/AirPlanJ/build/cl...
...asses')
else
    flightsDirectory = ['C:\Documents and Settings\Jeff\My Documents\Dr...
...op Box\Analysis\Mesoscopic Flow Model\Computational Results\Extracted ...
...Flights By Flightlevel\FL' num2str(flightLevel) ' 050829\'];
    windDirectory = 'C:\Documents and Settings\Jeff\My Documents\Java P...
...rojects\AirPlanJ\data\wind\';
    addpath('C:\Documents and Settings\Jeff\My Documents\Drop Box\Analy...
...sis\Mesoscopic Flow Model\MATLAB Separation\');
    addpath('C:\Documents and Settings\Jeff\My Documents\Drop Box\Analy...
...sis\Mesoscopic Flow Model\MATLAB Java Separation Simulation\');
    javaclasspath('C:\\Documents and Settings\\jeff\\My Documents\\Java...
... Projects\\AirPlanJ\\build\\classes\\')
end
flightidx = csvread([flightsDirectory 'flightidx.csv']);
coordinates = csvread([flightsDirectory 'coordinates.csv']);
x0ZWindData = csvread([windDirectory '0ZWindData.csv']);
x600ZWindData = csvread([windDirectory '600ZWindData.csv']);
x1200ZWindData = csvread([windDirectory '1200ZWindData.csv']);
x1800ZWindData = csvread([windDirectory '1800ZWindData.csv']);
disp(['Interpolating wind at flight level ' num2str(flightLevel) '...']...
...)
[x0ZWindDataAtFL] = interpolate_wind_at_altitude(x0ZWindData, flightLev...
...el);
[x600ZWindDataAtFL] = interpolate_wind_at_altitude(x600ZWindData, fligh...
```

```
...tLevel);
[x1200ZWindDataAtFL] = interpolate_wind_at_altitude(x1200ZWindData, fli...
...ghtLevel);
[x1800ZWindDataAtFL] = interpolate_wind_at_altitude(x1800ZWindData, fli...
...ghtLevel);
disp('Azimuth calculations...')
[coordinatesWithAzimuth] = calculate_azimuth(flightidx, coordinates);
disp('Interpolating position at 1 minute intervals...')
[interFlightidx, interCoordinates] = position_interpolation(...
    flightidx, coordinatesWithAzimuth);
disp('Creating a coordinate vector sorted by time...')
[interTimeIdx, interCoordinatesByTime] = time_indices(interCoordinates)...
...;
disp('Adding sector info to coordinates...')
[interCoordinatesByTime2] = ...
    add_sector_info_to_coordinates(interCoordinatesByTime);
disp('Performing the separation search...')
dummyCenterLat = [47; 25; 25; 47];
dummyCenterLon = [-105; -105; -63; -63];
calculateMITSeparation = false;
[separationMIT, separationNotMIT] = separation_search(interFlightidx, ....
.....
    interCoordinates, interTimeIdx, interCoordinatesByTime2, flightLeve...
...l,...
    x0ZWindDataAtFL, x600ZWindDataAtFL, x1200ZWindDataAtFL, x1800ZWindD...
...ataAtFL,...
    dummyCenterLat, dummyCenterLon, calculateMITSeparation);
toc
```

MATLAB Supporting Script for Aircraft Separation Calculations that Adds Wind Speed and Direction Information (interpolate_wind_at_altitude.m)

This script supports separation_main.m by adding wind information.

```
% Linear interpolation of wind data at altitude.
function [xNZWindDataAtFL] = interpolate_wind_at_altitude(xNZWindData, ...
...flightLevel)
% Find the flight level below and above
FL_below = 0;
FL_below_idx = 0;
FL_above = 999;
for i=1:9
    if and(xNZWindData(i,3) > FL_below, xNZWindData(i,3) < flightLevel)...
...       FL_below = xNZWindData(i,3);
       FL_below_idx = i;
    end

    if and(xNZWindData(i,3) < FL_above, xNZWindData(i,3) > flightLevel)...
...       FL_above = xNZWindData(i,3);
    end
end
len = length(xNZWindData)/9;
xNZWindDataAtFL = zeros(len,5);
for i=1:len
   idx_below = FL_below_idx + (i-1)*9;
   idx_above = idx_below + 1;

   % Set lat/lon
   xNZWindDataAtFL(i,1) = xNZWindData(idx_below,1);
   xNZWindDataAtFL(i,2) = xNZWindData(idx_below,2);

   % Set flight level
   xNZWindDataAtFL(i,3) = flightLevel;

   % Interpolate wind direction
   xNZWindDataAtFL(i,4) = interp1([FL_below FL_above], ...
       [xNZWindData(idx_below,4) xNZWindData(idx_above,4)], flightLevel...
...);

   % Interpolate wind speed
   xNZWindDataAtFL(i,5) = interp1([FL_below FL_above], ...
       [xNZWindData(idx_below,5) xNZWindData(idx_above,5)], flightLevel...
...);
end
```

MATLAB Supporting Script for Aircraft Separation Calculations that Adds Azimuth Information to the Radar Track (calculate_azimuth.m)

This script supports separation_main.m by adding azimuth information to the flight track data.

```
% Calculation of azimuth for flights with more than 3 waypoints
function [coordinatesWithAzimuth] = calculate_azimuth(flightidx, coordi...
...nates)
% Copy index,lat,lon,time,groundspeed from coordinates
coordinatesWithAzimuth = coordinates;
%
for i=1:length(flightidx)
    % Check if enough waypoints - if not then set azimuth to -1
    if flightidx(i,3)-flightidx(i,2) < 2
       coordinatesWithAzimuth(flightidx(i,2):flightidx(i,3),6) = -1;
       continue;
    end

    % Create a copy of the data
    waypoints = coordinatesWithAzimuth(flightidx(i,2):flightidx(i,3),:)...
...;

    % Calculate azimuth for first point between first and third points
    waypoints(1,6) = azimuth(waypoints(1,2),waypoints(1,3),...
        waypoints(3,2), waypoints(3,3));


    % Calculate azimuth for other points
    l_waypoints = length(waypoints(:,1));
    waypoints(2:l_waypoints-1,6) = azimuth(waypoints(1:l_waypoints-2,2)...
...,...
        waypoints(1:l_waypoints-2,3),waypoints(3:l_waypoints,2),...
        waypoints(3:l_waypoints,3));

    % Calculate azimuth for last point
    waypoints(l_waypoints,6) = azimuth(waypoints(l_waypoints-2,2),...
        waypoints(l_waypoints-2,3),waypoints(l_waypoints,2),...
        waypoints(l_waypoints,3));

    % Copy the data back
    coordinatesWithAzimuth(flightidx(i,2):flightidx(i,3),6) = ...
        waypoints(:,6);
end
```

MATLAB Supporting Script for Aircraft Separation Calculations by Interpolating Coordinates at a Consistent Time (position_interpolation.m)

This script supports separation_main.m by interpolating coordinates to create a consistent time.

```
% This function runs a linear interpolation to get location at 1 minute...
...% (exact) intervals.
%
% Workspace Variables
% coordinates from 'coordinates.csv'
% flightidx from 'flightidx.csv'
% Data:
% interCoordinates: Index, Latitude, Longitude, Time (nearest minute),
% Ground Speed, Azimuth, Aircraft Alias, Preferred Airspeed, Flight Ind...
...ex,
% Flight Level
%
% interFlightidx: Flight Index, Start Index, End Index, Start Time, End...
...% Time
function [interFlightidx, interCoordinates] = position_interpolation(.....
....
    flightidx, coordinatesWithAzimuth)
% Number of flights
numberFlights = length(flightidx);
% Calculate total number of flight locations after linear interpolation...
...numberTracks = sum(flightidx(:,5)) - sum(flightidx(:,4)) + numberFlight...
...s;
% Flight index with interpolation
interFlightidx = zeros(numberFlights, 5);
% Coordinates with interpolation
interCoordinates = zeros(numberTracks, 10);
% Loop through the flights
k=0;
for i=1:numberFlights
   startIdx = flightidx(i,2);
   endIdx = flightidx(i,3);

   flight_lat = coordinatesWithAzimuth(startIdx:endIdx, 2);
   flight_lon = coordinatesWithAzimuth(startIdx:endIdx, 3);
   flight_tm  = coordinatesWithAzimuth(startIdx:endIdx, 4);
   flight_az  = coordinatesWithAzimuth(startIdx:endIdx, 6);
   flight_tas = coordinatesWithAzimuth(startIdx:endIdx, 8);
   flight_fl  = coordinatesWithAzimuth(startIdx:endIdx, 10);

   % Ground speed
   flight_tm_gs = coordinatesWithAzimuth(startIdx:endIdx,4);
   flight_gs = coordinatesWithAzimuth(startIdx:endIdx,5);
```

```
   l_flight_gs = length(flight_gs);
   for j=0:l_flight_gs-1
       idx_j = l_flight_gs - j;
       if(flight_gs(idx_j) == -1)
          flight_gs(idx_j) = [];
          flight_tm_gs(idx_j) = [];
       end
   end

   startTm = flightidx(i,4);
   endTm = flightidx(i,5);

   interFlightidx(i,1) = i;
   interFlightidx(i,2) = k+1;
   interFlightidx(i,4) = startTm;
   interFlightidx(i,5) = endTm;

   for j=startTm:endTm
       k=k+1;
       interCoordinates(k,1) = k;
       try
         if endIdx > startIdx % i.e. more than one point on the traject...
...ory
            interCoordinates(k,2) = interp1(flight_tm, flight_lat, j, '...
...linear');
            interCoordinates(k,3) = interp1(flight_tm, flight_lon, j, '...
...linear');
            interCoordinates(k,6) = interp1(flight_tm, flight_az,  j, '...
...linear');
            interCoordinates(k,8) = interp1(flight_tm, flight_tas, j, '...
...linear');
            interCoordinates(k,10)= interp1(flight_tm, flight_fl,  j, '...
...linear');

            % Test for failure of interpolation
            if(isnan(interCoordinates(k,2)))
                interCoordinates(k,2) = flight_lat(1);
                interCoordinates(k,3) = flight_lon(1);
                interCoordinates(k,6) = flight_az(1);
                interCoordinates(k,8) = flight_tas(1);
                interCoordinates(k,10)= flight_fl(1);
            end
         else
            interCoordinates(k,2) = flight_lat;
            interCoordinates(k,3) = flight_lon;
            interCoordinates(k,6) = flight_az;
            interCoordinates(k,8) = flight_tas;
```

```
            interCoordinates(k,10)= flight_fl;
        end

        % Calculations for ground speed
        if length(flight_gs) > 1
            interCoordinates(k,5) = interp1(flight_tm_gs, flight_gs, j...
..., 'linear', 'extrap');
        elseif isempty(flight_gs)
            interCoordinates(k,5) = -1;
        else
            interCoordinates(k,5) = flight_gs;
        end

        % Aircraft performance is the same for all points
        interCoordinates(k,7) = coordinatesWithAzimuth(startIdx,7);

        % Flight index is the same for all flights
        interCoordinates(k,9) = coordinatesWithAzimuth(startIdx,9);

       catch
        disp('i = ');
        disp(i);
        disp('j = ');
        disp(j);
        toc;
        return;
       end
       interCoordinates(k,4) = j;
   end

   interFlightidx(i,3) = k;

end
save interCoordinates.mat interCoordinates
save interFlightidx.mat interFlightidx
```

MATLAB Supporting Script for Aircraft Separation Calculations that Adds Time Indices (time_indices.m)

This script supports separation_main.m by adding time indices so that all flight positions during a time interval can be analyzed.

```
% This script sorts the coordinates by time index:
%   interCoordinatesByTime: Index, Latitude, Longitude, Time, GroundSpe...
...ed,
%   Azimuth, Aircraft Alias, Preferred Airspeed, Flight Index, Flight L...
...evel
% Then sets up a time index:
%   interTimeIdx: Time, # Points, Start Index, End Index
function [interTimeIdx, interCoordinatesByTime] = time_indices(interCoo...
...rdinates)
% Copy of the coordinates sorted by flight
interCoordinatesByTime = interCoordinates;
% Sort by time
interCoordinatesByTime = sortrows(interCoordinatesByTime,4);
% Minimum time
minTime = min(interCoordinatesByTime(:,4));
% Maximum time
maxTime = max(interCoordinatesByTime(:,4));
% Create time index matrix
tmDuration = maxTime-minTime+1;
interTimeIdx = zeros(tmDuration-1,4);
% Record time
% Data for interTimeIdx: Time; # Points; Start Index; End Index
% Indices refer to interCoordinatesByTime
for i=0:tmDuration-1
   % Time starts at 0; Index starts at 1
   interTimeIdx(i+1,1)=i;
end
for j=1:length(interCoordinatesByTime)
    % Time starts at 0; Index starts at 1
    tmIdx = interCoordinatesByTime(j,4)+1;

    % Update the number of points
    numPts = interTimeIdx(tmIdx,2);
    interTimeIdx(tmIdx,2) = numPts + 1;

    % Start index
    startIdx = interTimeIdx(tmIdx,3);
    if startIdx == 0
        interTimeIdx(tmIdx,3) = j;
    end;

    % End index
```

```
    endIdx = interTimeIdx(tmIdx,4);
    if j > endIdx
        interTimeIdx(tmIdx,4) = j;
    end

end
```

MATLAB Supporting Script for Aircraft Separation Calculations that Adds the Sector the Flight Position Occupys (add_sector_info_to_coordinates.m)

This script supports separation_main.m by adding the sector that this flight radar track position occupys.

```
% This script adds sector information to coordinates.
%   interCoordinatesByTime2: Index, Latitude, Longitude, Time, GroundSp...
...eed,
%   Azimuth, Aircraft Alias, Preferred Airspeed, Flight Index, Flight L...
...evel,
%   Sector Idx, Sector Floor FL, Sector Ceiling FL
function [interCoordinatesByTime2] = ...
    add_sector_info_to_coordinates(interCoordinatesByTime)
% m = # rows; n = # columns
m = size(interCoordinatesByTime,1);
% Create the coordinate vector with sector info
sector_zeros = zeros(m,3);
interCoordinatesByTime2 = [interCoordinatesByTime sector_zeros];
latitude = interCoordinatesByTime2(:,2);
longitude= interCoordinatesByTime2(:,3);
% Load info from AirPlanJ
import AirPlanJ.*
APData.Initialize;
Settings.defaultSettings;
Settings.readRadarTrackOnly = false;
FAPio.readData;
centers = APData.airspace.getCenterKeys;
sector_idx = 0;
while(centers.hasMoreElements)
    currentCenterName = centers.nextElement;
    centerObject = APData.airspace.getCenter(currentCenterName);

    sectors = centerObject.getSectorElements;
    while(sectors.hasMoreElements)
        sectorObject = sectors.nextElement;

        % Don't consider low altitude sectors
        if(sectorObject.getCeiling < 180)
            continue;
        end
        % Problematic oceanic sectors
        if(strcmp(sectorObject.getName,'ZNY8120'))
            continue;
        elseif(strcmp(sectorObject.getName, 'ZNY8110'))
            continue;
        elseif(strcmp(sectorObject.getName, 'ZNY8101'))
            continue;
```

```
        end

        disp(sectorObject.getName)
        sector_idx = sector_idx + 1;
        disp(['Index = ' num2str(sector_idx)])

        for i = 0:sectorObject.noModules-1
            moduleObject = sectorObject.getModule(i);

            sectorLatitude = zeros(moduleObject.noNodes,1);
            sectorLongitude= sectorLatitude;

            idx = 1;
            for j=0:moduleObject.noNodes-1
                pt = moduleObject.getNode(j);
                sectorLatitude(idx) = double(pt.getLatitude);
                sectorLongitude(idx) = double(pt.getLongitude);
                idx = idx + 1;
            end

            moduleFloor = double(moduleObject.getFloor);
            moduleCeiling=double(moduleObject.getCeiling);

            IN = inpolygon(latitude,longitude,sectorLatitude,sectorLong...
...itude);

            for j=1:length(IN)
                % Check that altitude is between module floor and ceili...
...ng
                FL = interCoordinatesByTime2(j,10);
                if(IN(j) == 1 && FL >= moduleFloor && FL <= moduleCeili...
...ng)
                   interCoordinatesByTime2(j,11) = sector_idx;
                   interCoordinatesByTime2(j,12) = moduleFloor;
                   interCoordinatesByTime2(j,13) = moduleCeiling;
                end
            end
        end
    end
end
```

MATLAB Supporting Script for Aircraft Separation Calculations that Performs the Actual Search (separation_search.m)

This script supports separation_main.m by performing the search for the closest aircraft.

```
% This script finds the closest nearby flight
%
% Note: The warning "One or more points did not converge."
% Happens when performing the dsearch to remove the point
% outside the triangulation.
function[separationMIT, separationNotMIT] = separation_search(interFlig...
...htidx, ...
    interCoordinates, interTimeIdx, interCoordinatesByTime, flightLevel...
...,...
    x0ZWindDataAtFL, x600ZWindDataAtFL, x1200ZWindDataAtFL, x1800ZWindD...
...ataAtFL, ...
    centerLat, centerLon, calculateMITSeparation)
% Array to hold separation values:
% (Separation (nm); Groundspeed; In Smaller Polygon (Boolean);
% Flight Index; Time; GSi; BADAi(TAS); AZi; GSk; BADAk(TAS); AZiTOk; AZ...
...Diff;
% Time To Collision; Alpha; Airspeedi; Airspeedk; Flight Index 2;
% Sector Index; Sector Floor; Sector Ceiling)
separationNotMIT = zeros(length(interCoordinates),20);
k=1;
% Array to hold MIT separation values and number of flights using
% separation.
% separationMIT: (# conflicting flights using separation; separation;
%                    groundspeed; BADA TAS Lead; Observed TAS Lead;
%                    Lead TAS Corrected for wind;
%                    BADA TAS Following; Observed TAS Following;
%                    Following TAS Corrected for wind)
separationMIT = zeros(length(interCoordinates),9);
% Number of times separation less than 3nm
countLessThan3nm = 0;
countLessThan3nmMIT = 0;
% Get smaller center polygon for verification purposes
[centerLat2, centerLon2] = tmp_function(centerLat, centerLon);
for n=1:length(interFlightidx)

    %n
    for flightIndex = interFlightidx(n,2):interFlightidx(n,3)
        %flightIndex
        % First point in the flight trajectory
        %flightIndex = interFlightidx(1,2);
        flightTime = interCoordinates(flightIndex,4);
        % Extract flight coordinates at flight time
        lat_i = interCoordinates(flightIndex,2);
```

```
        lon_i = interCoordinates(flightIndex,3);
        gs_i  = interCoordinates(flightIndex,5);    % Groundspeed
        az_i  = interCoordinates(flightIndex,6);    % Azimuth
        tas_i = interCoordinates(flightIndex,8);    % TAS [kts]

        % Set groundspeed for flight
        separationNotMIT(k,2) = interCoordinates(flightIndex,5);
        separationMIT(k,3) = interCoordinates(flightIndex,5);

        % Record flight and time and GSi,BADAi,AZi
        separationNotMIT(k,4) = interFlightidx(n,1);
        separationNotMIT(k,5) = flightTime;
        separationNotMIT(k,6) = gs_i;
        separationNotMIT(k,7) = tas_i;
        separationNotMIT(k,8) = az_i;
        % Extract other flights
        % - Recall that times are 0 based
        startTimeIdx = interTimeIdx(flightTime+1,3);
        endTimeIdx = interTimeIdx(flightTime+1,4);
        lat = interCoordinatesByTime(startTimeIdx:endTimeIdx,2);
        lon = interCoordinatesByTime(startTimeIdx:endTimeIdx,3);
        gs  = interCoordinatesByTime(startTimeIdx:endTimeIdx,5);
        az  = interCoordinatesByTime(startTimeIdx:endTimeIdx,6);
        tas = interCoordinatesByTime(startTimeIdx:endTimeIdx,8);
        fl_idx = interCoordinatesByTime(startTimeIdx:endTimeIdx,9);
        alt = interCoordinatesByTime(startTimeIdx:endTimeIdx,10);   % A...
...ltitude
        sect_idx = interCoordinatesByTime(startTimeIdx:endTimeIdx,11); ...
...% Sector Index
        floor = interCoordinatesByTime(startTimeIdx:endTimeIdx,12); % S...
...ector Floor
        ceiling = interCoordinatesByTime(startTimeIdx:endTimeIdx,13); %...
... Sector Ceiling

        % Check if too few lat/lon points
        % This condition should not occur normally
        if(length(lat) < 4)
            separationNotMIT(k,1) = 999;
            separationMIT(k,2) = 999;
            k=k+1;
            continue;
        %elseif(length(lat) == 2)
        %    distDegrees = distance(lat(1),lon(1),lat(2),lon(2));
        %    distNM = deg2nm(distDegrees);
        %    separationNotMIT(k) = distNM;
        %    k=k+1;
        %    continue;
```

```
        end
        % The closest coordinates is the flight and must be removed
        % i.e. remove point with distance = 0.
        %TRI = delaunay(lat, lon);
        %K = dsearch(lat,lon,TRI,lat_i,lon_i);
        K = 0;
        for Ki = 1:length(lat)
            if(fl_idx(Ki) == interFlightidx(n,1))
                K = Ki;
                break;
            end
        end
        if(inpolygon(lon(K), lat(K), centerLon2, centerLat2))
           separationNotMIT(k,3) = 1;
        end
        lat(K) = [];
        lon(K) = [];
        az(K)  = [];
        gs(K)  = [];
        tas(K) = [];
        fl_idx(K) = [];
        flight_altitude = alt(K); alt(K) = [];
        flight_sector = sect_idx(K); sect_idx(K) = [];
        flight_floor = floor(K); floor(K) = [];
        flight_ceiling = ceiling(K); ceiling(K) = [];
        separationNotMIT(k,18) = flight_sector;
        separationNotMIT(k,19) = flight_floor;
        separationNotMIT(k,20) = flight_ceiling;

        % Remove flights outside of the altitude range
        length_lat = size(lat,1);
        for j=1:length_lat
           kj = length_lat - j + 1;
           if(alt(kj) < flight_floor || alt(kj) > flight_ceiling)
               lat(kj) = []; lon(kj) = []; az(kj) = []; gs(kj) = [];
               tas(kj) = []; fl_idx(kj) = []; alt(kj) = []; sect_idx(kj...
...) = [];
               floor(kj) = []; ceiling(kj) = [];
           end
        end

        % If sector info has been added only consider the closest fligh...
...t if
        % it is in the flight level range
%         closest_flight_altitude_range = 0;
%         while(closest_flight_altitude_range ~= 1)
%            TRI = delaunay(lat,lon);
```

```
%            K = dsearch(lat,lon,TRI,lat_i,lon_i);
%            if(alt(K) >= flight_floor && alt(K) <= flight_ceiling)
%                closest_flight_altitude_range = 1;
%            elseif length(lat) < 5
%                K = 1;
%                closest_flight_altitude_range = 1;
%            else
%                lat(K) = []; lon(K) = []; az(K) = []; gs(K) = [];
%                tas(K) = []; fl_idx(K) = []; alt(K) = []; sect_idx(K) ...
...= [];
%                floor(K) = []; ceiling(K) = [];
%            end
%          end
        % Check again if too few lat/lon points
        % This condition may occur for low altitude flight trajectory
        % locations
        if(length(lat) < 10)
            separationNotMIT(k,1) = 999;
            separationMIT(k,2) = 999;
            k=k+1;
            continue;
        end
        % Search again and find a new point
        TRI = delaunay(lat, lon);
        K = dsearch(lat,lon,TRI,lat_i,lon_i);
        % Calculate separation and record result
        distDegrees = distance(lat_i,lon_i,lat(K),lon(K));
        distNM = deg2nm(distDegrees);
        separationNotMIT(k,1) = distNM;

        separationNotMIT(k,9) = gs(K);
        separationNotMIT(k,10) = tas(K);
        separationNotMIT(k,11) = azimuth(lat_i,lon_i,lat(K),lon(K));

        % Calculate the difference in azimuths
        angle_a = deg2rad(az_i);
        angle_b = deg2rad(separationNotMIT(k,11));
        a = [sin(angle_a) cos(angle_a)];
        b = [sin(angle_b) cos(angle_b)];
        theta = acos(dot(a,b)/norm(a)/norm(b));
        separationNotMIT(k,12) = rad2deg(theta);

        % Calculate the time to collision (minutes)
        if gs_i > gs(K)
           separationNotMIT(k,13) = 60.0*distNM/(gs_i - gs(K));
        else
           separationNotMIT(k,13) = -1; % Following is slower than lead...
```

```
... aircraft
        end

        % Calculate alpha
        if tas_i > tas(K)
            alpha = (gs_i - gs(K))/(tas_i - tas(K));
        else
            alpha = -1;
        end
        separationNotMIT(k,14) = alpha;

        % Calculate airspeed
        if tas_i > 0
            separationNotMIT(k,15) = convert_groundspeed_to_airspeed(.....
....
                gs_i, az_i, flightTime, lat_i, lon_i, ...
                x0ZWindDataAtFL, x600ZWindDataAtFL, x1200ZWindDataAtFL,...
... x1800ZWindDataAtFL);
        else
           separationNotMIT(k,15) = -9999;
        end

        if tas(K) > 0
            separationNotMIT(k,16) = convert_groundspeed_to_airspeed(.....
....
                gs(K), az(K), flightTime, lat(K), lon(K), ...
                x0ZWindDataAtFL, x600ZWindDataAtFL, x1200ZWindDataAtFL,...
... x1800ZWindDataAtFL);
        else
           separationNotMIT(k,16) = -9999;
        end

        % Set conflicting flight index
        separationNotMIT(k,17) = fl_idx(K);



        % Check for problems
        if(distNM < 3)
            countLessThan3nm = countLessThan3nm + 1;
            disp(['n = ' num2str(n) '; flightIndex = ' num2str(flightIn...
...dex)])
        end

        %
        % Calculate MIT separation
        if calculateMITSeparation
```

```
            %
            % Initialize separation to 999
            separationMIT(k,2) = 999;
            lat_fl = interCoordinates(flightIndex:interFlightidx(n,3),2...
...);
            lon_fl = interCoordinates(flightIndex:interFlightidx(n,3),3...
...);
            % Check if there are too few points
            if length(lat_fl) < 4
                separationMIT(k,2) = 999;
                k=k+1;
                continue;
            end
            % Calculate length of the flight track
            distFlightTrack = deg2nm(distance(lat_fl(1),lon_fl(1), lat_...
...fl(length(lat_fl)), ...
                lon_fl(length(lon_fl))));
            % Create 3nm polygon around flight track
            [poly_lat, poly_lon] = flight_track_3nm_polygon(lat_fl,lon_...
...fl,3);
            while(length(lat) >= 4)
               % Find the closest point
               %disp(['Location = 4; n = ' num2str(n) '; flightIndex = ...
...' num2str(flightIndex)])
               TRI = delaunay(lat, lon);
               %disp(['Location = 5; n = ' num2str(n) '; flightIndex = ...
...' num2str(flightIndex)])
               K = dsearch(lat,lon,TRI,lat_i,lon_i);
               %disp(['Location = 6; n = ' num2str(n) '; flightIndex = ...
...' num2str(flightIndex)])
               distDegrees = distance(lat_i,lon_i,lat(K),lon(K));
               distNM = deg2nm(distDegrees);
               % Check to see if it is in the polygon
               in = inpolygon(lat(K),lon(K),poly_lat,poly_lon);
               % If not in polygon remove from consideration
               if not(in)
                  % Break if closest point is further away than the pol...
...ygon
                  % length
                  if(distNM > distFlightTrack + 3)
                      break;
                  end
                  lat(K) = []; lon(K) = []; az(K) = []; gs(K) = [];
                  tas(K) = []; fl_idx(K) = []; alt(K) = []; sect_idx(K)...
... = [];
                  floor(K) = []; ceiling(K) = [];
                  continue;
```

```
            end
            % If number of waypoints for conflicting < 3 (azimuth = ...
...-1)
            % or difference in azimuth is greater than 30 degrees
            % then mark as conflicting
            if az(K) == -1 || and(abs(az(K)-az_i) > 30,abs(az(K)-az_...
...i) < 330)
                separationMIT(k,1) = separationMIT(k,1) + 1;
                lat(K) = []; lon(K) = []; az(K) = []; gs(K) = [];
                tas(K) = []; fl_idx(K) = []; alt(K) = []; sect_idx(...
...K) = [];
                floor(K) = []; ceiling(K) = [];
                continue;
            end
            % Otherwise the flight is in trail
            separationMIT(k,2) = distNM;
            % Lead BADA TAS
            separationMIT(k,4) = tas(K);
            % Lead Observed TAS
            separationMIT(k,5) = gs(K);
            % Lead TAS Corrected for wind
            [separationMIT(k,6)] = convert_groundspeed_to_airspeed(....
.....
                gs(K), az(K), flightTime, lat(K), lon(K),...
                x0ZWindDataAtFL, x600ZWindDataAtFL, x1200ZWindDataAt...
...FL, x1800ZWindDataAtFL);
            % Following BADA TAS
            separationMIT(k,7) = tas_i;
            % Following Observed TAS
            separationMIT(k,8) = gs_i;
            % Following TAS Corrected for wind
            [separationMIT(k,9)] = convert_groundspeed_to_airspeed(....
.....
                gs_i, az_i, flightTime, lat_i, lon_i,...
                x0ZWindDataAtFL, x600ZWindDataAtFL, x1200ZWindDataAt...
...FL, x1800ZWindDataAtFL);
    %           if(or(k==59,k==200))
    %              disp(['k = :' num2str(k)])
    %              disp(['n = :' num2str(n)])
    %              disp(['fl_idx(K) = :' num2str(fl_idx(K))])
    %              disp(['az_i = :' num2str(az_i)])
    %              disp(['az(K) = :' num2str(az(K))])
    %              disp(' ')
    %           end
            % Check for problems
            if(distNM < 3)
                countLessThan3nmMIT = countLessThan3nmMIT + 1;
```

```
                    disp(['MIT: n = ' num2str(n) '; flightIndex = ' num2...
...str(flightIndex)])
                end
                % Exit the while loop after MIT separation found
                break;
            end

        end
        k=k+1;
    end
end
disp(['Observations less than 3 nm:    ' num2str(countLessThan3nm)])
disp(['MIT observations less than 3nm: ' num2str(countLessThan3nmMIT)])...
...;
save('separationMIT_FL350.mat', 'separationMIT');
save('separationNotMIT_FL350.mat', 'separationNotMIT');
csvwrite('separationMIT.csv', separationMIT);
csvwrite('separationNotMIT.csv', separationNotMIT);
```

MATLAB Script that Simulates Flights for the Purpose of Validating the Conflict Prediction Model (sector_separation_simulation2.m)

This script simulates flights through the New York Air Route Traffic Control Center (ARTCC) for the purpose of validating the conflict prediction model.

```
% Set path for java .class files
clear
tic
% Flight level under consideration. Filter sector modules below this fl...
...ight
% level.
flightLevel = 350;
% Use the same groundspeed for all flights [knots]
groundSpeed = 450;
% Map of New York
new_york = shaperead('usastatehi',...
'UseGeoCoords', true,...
'Selector',{@(name) strcmpi(name,'New York'), 'Name'});
plot(new_york.Lon, new_york.Lat)
% Load files
isMac = true;
%isMac = false;
%flightsDirectory = '';
%windDirectory = '';
if isMac
    %flightsDirectory = '/Users/antoniotrani/Documents/Jeff/AirPlanJ/da...
...ta/flights/';
    windDirectory = '/Users/antoniotrani/Documents/Jeff/AirPlanJ/data/w...
...ind/';
    addpath('/Users/antoniotrani/Documents/Jeff/Mesoscopic Flow Model/M...
...ATLAB Separation/');
    addpath('/Users/antoniotrani/Documents/Jeff/Mesoscopic Flow Model/L...
...inear Interpolation/');
    javaclasspath('/Users/antoniotrani/Documents/Jeff/AirPlanJ/build/cl...
...asses')
else
    %flightsDirectory = 'C:\\Documents and Settings\\jeff\\My Documents...
...\\Java Projects\\AirPlanJ\\data\\flights\\';
    windDirectory = 'C:\\Documents and Settings\\jeff\\My Documents\\Ja...
...va Projects\\AirPlanJ\\data\\wind\\';
    addpath('C:\\Documents and Settings\\jeff\\My Documents\\Drop Box\\...
...Analysis\\Mesoscopic Flow Model\\MATLAB Separation\\');
    addpath('C:\\Documents and Settings\\jeff\\My Documents\\Drop Box\\...
...Analysis\\Mesoscopic Flow Model\\Linear Interpolation\\');
    javaclasspath('C:\\Documents and Settings\\jeff\\My Documents\\Java...
... Projects\\AirPlanJ\\build\\classes\\')
end
```

```
import AirPlanJ.*
% Recreate the main routine
APData.Initialize;
Settings.defaultSettings;
FAPio.readData;
% Get list of centers
centers = APData.airspace.getCenterKeys;
% Get first center
currentCenterName = centers.nextElement;
centerObject = APData.airspace.getCenter(currentCenterName);
% Storage of center lat lon
centerLat = zeros(1000,1);
centerLon = zeros(1000,1);
centerLatLonLength = 0;
% Get the first sector
sectors = centerObject.getSectorElements;
while(sectors.hasMoreElements)
    sectorObject = sectors.nextElement;

    if(sectorObject.getCeiling < flightLevel)
        continue;
    end
    if(strcmp(sectorObject.getName,'ZNY8120'))
        continue;
    elseif(strcmp(sectorObject.getName, 'ZNY8110'))
        continue;
    elseif(strcmp(sectorObject.getName, 'ZNY8101'))
        continue;
    end
    % Number of sector modules
    numberSectorModules = sectorObject.noModules;
    numberModulePoints = 0;
    for i=1:numberSectorModules
        moduleIndex = i-1;
        moduleObject = sectorObject.getModule(moduleIndex);

        % Don't consider sector modules below flight level
        if(moduleObject.getCeiling < flightLevel)
            continue;
        end

        numberModulePoints = numberModulePoints + moduleObject.noNodes;...
...     end
    % Create vector to store sector points
    sectorLatitude = zeros(numberModulePoints,1);
    sectorLongitude= zeros(numberModulePoints,1);
    %
```

```
    idx = 1;
    for i=1:numberSectorModules
       moduleIndex = i-1;
       moduleObject = sectorObject.getModule(moduleIndex);

       % Don't consider sector modules below flight level
       if(moduleObject.getCeiling < flightLevel)
           continue;
       end

       for j=1:moduleObject.noNodes
          pt = moduleObject.getNode(j-1);
          sectorLatitude(idx) = double(pt.getLatitude);
          sectorLongitude(idx) = double(pt.getLongitude);
          idx = idx + 1;
       end
    end
    % Get the airway locations
    airways = APData.airspace.getJetRouteKeys;
    airwayIdx = 1;
    intersectingAwyLat = zeros(125,25);
    intersectingAwyLon = zeros(125,25);
    lengthAwy = 0;
    % Get airways
    while(airways.hasMoreElements)
        currentAirwayName = airways.nextElement;
        airwayObject = APData.airspace.getJetRoute(currentAirwayName);
        numberAirwayPoints = airwayObject.noPoints;
        airwayLatitude = zeros(numberAirwayPoints,1);
        airwayLongitude = zeros(numberAirwayPoints,1);
        idx = 1;
        for i=1:numberAirwayPoints
            pt = airwayObject.getPoint(i-1);
            airwayLatitude(idx) = double(pt.getLatitude);
            airwayLongitude(idx) = double(pt.getLongitude);
            idx = idx + 1;
        end

        % Check for intersection of airway and sector
        [airwayLatitude airwayLongitude inSector] = airway_in_sector(.....
....
            airwayLatitude, airwayLongitude, sectorLatitude, sectorLong...
...itude);
        if(not(inSector))
            continue;
        end
```

```
        % Check that all airway points are in the sector
        airwayPointNotInSector = 0;
        if(length(airwayLatitude) > 2)
            for i=2:length(airwayLatitude)-1
                if(not(inpolygon(airwayLongitude(i), airwayLatitude(i),...
......
                        sectorLongitude, sectorLatitude)))
                   airwayPointNotInSector = i;
                   break;
                end
            end
        end
        % Case all airway points in sector boundary
        if(airwayPointNotInSector == 0)
            intersectingAwyLat(1:length(airwayLatitude),airwayIdx) = ai...
...rwayLatitude;
            intersectingAwyLon(1:length(airwayLongitude),airwayIdx) = a...
...irwayLongitude;
            lengthAwy(airwayIdx) = length(airwayLatitude);
            airwayIdx = airwayIdx + 1;
        else % At least one airway point outside sector boundary
            % Break the airway into two segments
            airwayLatitude1 = airwayLatitude(1:airwayPointNotInSector);...
...            airwayLongitude1 = airwayLongitude(1:airwayPointNotInSector...
...);
            [airwayLatitude1 airwayLongitude1 inSector1] = airway_in_se...
...ctor(...
                airwayLatitude1, airwayLongitude1, sectorLatitude, sect...
...orLongitude);
            lengthAwy(airwayIdx) = length(airwayLatitude1);
            airwayIdx = airwayIdx + 1;

            airwayLatitude2 = airwayLatitude(airwayPointNotInSector:len...
...gth(airwayLatitude));
            airwayLongitude2 = airwayLongitude(airwayPointNotInSector:l...
...ength(airwayLongitude));
            [airwayLatitude2 airwayLongitude2 inSector2] = airway_in_se...
...ctor(...
                airwayLatitude2, airwayLongitude2, sectorLatitude, sect...
...orLongitude);
            lengthAwy(airwayIdx) = length(airwayLatitude2);
            airwayIdx = airwayIdx + 1;
        end
    end
    % Now plot the results
%     hold on
%     plot(sectorLongitude, sectorLatitude, 'k');
```

```
%
%     for i=1:airwayIdx-1
%        hold on;
%
%        plot(intersectingAwyLon(1:lengthAwy(i),i), ...
%             intersectingAwyLat(1:lengthAwy(i),i), 'b');
%     end
    %pause
    % Append sector lat lon to center lat lon
    sectorLength = length(sectorLatitude);
    centerLat(centerLatLonLength+1:centerLatLonLength+sectorLength) = s...
...ectorLatitude;
    centerLon(centerLatLonLength+1:centerLatLonLength+sectorLength) = s...
...ectorLongitude;
    centerLatLonLength = centerLatLonLength + sectorLength;

end
% Expected sector distribution (intensity)
% ==> Intensity = ((lambda1*traversal_time1 +
% lambda2*traversal_time2)/area)
% Consider a random case
% Create a convex hull around the ZNY ARTCC boundary
centerLat = centerLat(1:centerLatLonLength);
centerLon = centerLon(1:centerLatLonLength);
K = convhull(centerLon, centerLat);
centerLat = centerLat(K);
centerLon = centerLon(K);
hold on
plot(centerLon, centerLat, 'r');
% Consider 90 flights in 2 hours
% 90/120 = 0.75 flights/minute
simulationFlights = 500; % Number of flights in the simulation
averageInterarrivalTime = 1.25;
averageArrivalRate = 1/averageInterarrivalTime;
startTime = 0;
firstFlightExit = -1;
lastFlightEnter = -1;
randomFlightLocations = [];
sectorTraversalTime = zeros(90,1);
for i=1:simulationFlights
    % Randomly select an entering face and exiting face using
    % random permutations.
    [enterFaceLat enterFaceLon exitFaceLat exitFaceLon enterToExitAzimu...
...th] = ...
        random_face_select(centerLat, centerLon);
```

```
    % Randomly generate starting time (sector entry time)
    flightStartTime = -1*averageInterarrivalTime*log(rand);
    startTime = startTime + flightStartTime;
    flightStartTime = startTime;
    if(i == simulationFlights)
       lastFlightEnter = floor(flightStartTime);
    end

    % Get distance
    traversalDistance = distance(enterFaceLat, enterFaceLon, exitFaceLa...
...t,...
        exitFaceLon);
    traversalDistanceNM = deg2nm(traversalDistance);
    traversalTimeMinutes = 60*traversalDistanceNM/groundSpeed;
    flightEndTime = traversalTimeMinutes + flightStartTime;
    sectorTraversalTime(i) = traversalTimeMinutes;
    if(i == 1)
        firstFlightExit = ceil(flightEndTime);
    end

    % Interpolate location at 1 minute intervals
    interpolationTimes = ceil(flightStartTime):floor(flightEndTime);
    interpolationLatitude = interp1([flightStartTime flightEndTime], .....
....
        [enterFaceLat exitFaceLat], interpolationTimes);
    interpolationLongitude = interp1([flightStartTime flightEndTime], ....
.....
        [enterFaceLon exitFaceLon], interpolationTimes);

    randomFlightLocations_i = [interpolationLatitude' interpolationLong...
...itude' ...
        interpolationTimes'];

    randomFlightLocations_i(:,4) = groundSpeed;
    randomFlightLocations_i(:,5) = enterToExitAzimuth;
    randomFlightLocations_i(:,6) = 0;
    randomFlightLocations_i(:,7) = groundSpeed;
    randomFlightLocations_i(:,8) = i;

    randomFlightLocations = [randomFlightLocations; randomFlightLocatio...
...ns_i];

    hold on
    plot([enterFaceLon exitFaceLon], [enterFaceLat exitFaceLat], 'k')
end
% Remove locations before first flight exit and after last flight enter...
...l_fl = length(randomFlightLocations);
```

```
for i=1:l_fl
    idx = l_fl - i + 1;
    if(or(randomFlightLocations(idx,3) < firstFlightExit, ...
            randomFlightLocations(idx,3) > lastFlightEnter))
       randomFlightLocations(idx,:) = [];
    end
end
% interCoordinates: Index, Latitude, Longitude, Time (nearest minute),
% Ground Speed, Azimuth, Aircraft Alias, Preferred Airspeed, Flight Ind...
...ex
index = 1:length(randomFlightLocations);
index = index';
interCoordinates = [index randomFlightLocations];
% interFlightidx: Flight Index, Start Index, End Index, Start Time, End...
...% Time
interFlightidx = zeros(simulationFlights, 5);
idx2 = 1:simulationFlights;
idx2 = idx2';
interFlightidx(:,1) = idx2;
for i=1:length(interCoordinates)
   flightIndex_ = interCoordinates(i,9);
   index_ = interCoordinates(i,1);
   time_ = interCoordinates(i,4);
   if(or(interFlightidx(flightIndex_,2) == 0, interFlightidx(flightInde...
...x_,2) > index_))
       interFlightidx(flightIndex_,2) = index_;
   end

   if(or(interFlightidx(flightIndex_,3) == 0, interFlightidx(flightInde...
...x_,3) < index_))
       interFlightidx(flightIndex_,3) = index_;
   end

   if(or(interFlightidx(flightIndex_,4) == 0, interFlightidx(flightInde...
...x_,4) > time_))
       interFlightidx(flightIndex_,4) = time_;
   end

   if(or(interFlightidx(flightIndex_,5) == 0, interFlightidx(flightInde...
...x_,5) < time_))
       interFlightidx(flightIndex_,5) = time_;
   end
end
% Remove flights with no points
l_interFlightidx = length(interFlightidx);
for i=1:l_interFlightidx
   idx = l_interFlightidx - i + 1;
```

```
   if(or(interFlightidx(idx,4) == 0, interFlightidx(idx,5) == 0))
      interFlightidx(idx,:) = [];
   end
end
% Subtract minimum time so that time is zero-based (for 'time_indices'
% function)
interCoordinates(:,4) = interCoordinates(:,4) - firstFlightExit;
interFlightidx(:,4) = interFlightidx(:,4) - firstFlightExit;
interFlightidx(:,5) = interFlightidx(:,5) - firstFlightExit;
x0ZWindData = csvread([windDirectory '0ZWindData.csv']);
x600ZWindData = csvread([windDirectory '600ZWindData.csv']);
x1200ZWindData = csvread([windDirectory '1200ZWindData.csv']);
x1800ZWindData = csvread([windDirectory '1800ZWindData.csv']);
disp('Creating a coordinate vector sorted by time...')
[interTimeIdx, interCoordinatesByTime] = time_indices(interCoordinates)...
...;
disp('Performing the separation search...')
[separationMIT, separationNotMIT] = separation_search(interFlightidx, ....
.....
    interCoordinates, interTimeIdx, interCoordinatesByTime, flightLevel...
...,...
    x0ZWindData, x600ZWindData, x1200ZWindData, x1800ZWindData, centerL...
...at, centerLon);
% Calculate predicted separation
mean_t = mean(sectorTraversalTime);
center_area = areaint(centerLat, centerLon, almanac('earth','ellipsoid'...
...,'nauticalmiles'));
%averageInterarrivalTime
%expected_intensity = averageInterarrivalTime*mean_t/center_area;
expected_intensity = averageArrivalRate*mean_t/center_area;
average_separation = 32/(9*pi*sqrt(expected_intensity));
expected_min_separation = average_separation*(3/pi)*(35/(48*pi)+pi/18);...
...disp(['Predicted min separation: ' num2str(expected_min_separation)])
% Calculate observed separation
disp(['Average min separation: ' num2str(mean(separationNotMIT(:,1)))]);...
...disp(['Minimum observed separation: ' num2str(min(separationNotMIT(:,1)...
...))])
disp(['Maximum observed separation: ' num2str(max(separationNotMIT(:,1)...
...))])
% Calculate predicted time in conflict
minimum_separation = 3; %nm
conflict_area = pi*minimum_separation^2;
expected_conflict_intensity = expected_intensity*conflict_area;
exp_lambda = exp(-1*expected_conflict_intensity);
%P_Conflict = 0;
%for i=2:10
%   P_i = exp_lambda*(expected_conflict_intensity^i)/factorial(i);
```

```
%   P_Conflict = P_Conflict + P_i*i;
%end
P_0 = exp_lambda*(expected_conflict_intensity^0)/factorial(0);
P_1 = exp_lambda*(expected_conflict_intensity^1)/factorial(1);
%P_Conflict = (1-P_0-P_1)/(1-P_0);
P_Conflict = 1 - P_0;
E_Conflict = P_Conflict*expected_intensity*center_area;
dt = lastFlightEnter - firstFlightExit+1;
expected_time_in_conflict = dt*E_Conflict;
disp(['Predicted time in conflict: ' num2str(expected_time_in_conflict)...
...])
% Actual time in conflict
total_time_in_conflict = 0;
for i=1:length(separationNotMIT)
   if(separationNotMIT(i,1) <= minimum_separation)
      total_time_in_conflict = total_time_in_conflict + 1;
   end
end
% Need to divide by 2 since two flights/conflict
%total_time_in_conflict = total_time_in_conflict/2;
disp(['Observed time in conflict: ' num2str(total_time_in_conflict)])
% Calculate number of points in the inner polygon and outer polygon
num_inner_polygon = 0;
num_outer_polygon = 0;
for i=1:length(separationNotMIT)
   if(separationNotMIT(i,3) == 0)
      num_outer_polygon = num_outer_polygon + 1;
   else
      num_inner_polygon = num_inner_polygon + 1;
   end
end
disp(['Number of flights in the inner polygon: ' num2str(num_inner_poly...
...gon)])
disp(['Number of flights in the outer polygon: ' num2str(num_outer_poly...
...gon)])
% Calculate number of points in the inner polygon and outer polygon
num_inner_polygon = 0;
num_outer_polygon = 0;
separation_inner_polygon = 0;
separation_outer_polygon = 0;
conflict_t_inner_polygon = 0;
conflict_t_outer_polygon = 0;
for i=1:length(separationNotMIT)
   if(separationNotMIT(i,3) == 0)
      num_outer_polygon = num_outer_polygon + 1;
      separation_outer_polygon = separation_outer_polygon + separationN...
...otMIT(i,1);
```

```
      if(separationNotMIT(i,1) < 3)
          conflict_t_outer_polygon = conflict_t_outer_polygon + 1;
      end
   else
      num_inner_polygon = num_inner_polygon + 1;
      separation_inner_polygon = separation_inner_polygon  + separation...
...NotMIT(i,1);
      if(separationNotMIT(i,1) < 3)
         conflict_t_inner_polygon = conflict_t_inner_polygon + 1;
      end
   end
end
disp(['Number of flights in the inner polygon: ' num2str(num_inner_poly...
...gon)])
disp(['Number of flights in the outer polygon: ' num2str(num_outer_poly...
...gon)])
disp(['Average min separation in inner polygon: ' num2str(separation_in...
...ner_polygon/num_inner_polygon)])
disp(['Average min separation in outer polygon: ' num2str(separation_ou...
...ter_polygon/num_outer_polygon)])
disp(['Conflict time in inner polygon: ' num2str(conflict_t_inner_polyg...
...on)])
disp(['Conflict time in outer polygon: ' num2str(conflict_t_outer_polyg...
...on)])
toc
```

MATLAB Script that Supports the Simulation (airway_in_sector.m)

This script supports sector_separation_simulation2.m.

```
% Returns the portion of an airway that is contained in a sector.
function [airwayLatitude airwayLongitude inSector] = airway_in_sector(....
.....
    airwayLatitude, airwayLongitude, sectorLatitude, sectorLongitude)
inSector = false;
% Check for intersection
[lon_inter, lat_inter, ii_inter] = polyxpoly(airwayLongitude, ...
    airwayLatitude, sectorLongitude, sectorLatitude, 'unique');
if(isempty(lon_inter))
    return;
end
inSector = true;
% Create points that are only inside the sector or on the boundary
l_inter = length(ii_inter(:,1));
ii_start = ii_inter(1,1);
ii_end = ii_inter(l_inter,1);
if ii_start == ii_end % No points inside sector
   airwayLongitude = [lon_inter(1); lon_inter(l_inter)];
   airwayLatitude = [lat_inter(1); lat_inter(l_inter)];
else % At least one point inside sector
   airwayLongitude = [lon_inter(1); airwayLongitude(ii_start+1:ii_end);...
......
       lon_inter(l_inter)];
   airwayLatitude = [lat_inter(1); airwayLatitude(ii_start+1:ii_end);.....
....
       lat_inter(l_inter)];
end
```

MATLAB Script that Supports the Simulation (random_face_select.m)

This script supports sector_separation_simulation2.m.

```
% Random selection of a face
function [enterFaceLat enterFaceLon exitFaceLat exitFaceLon enterToExit...
...Azimuth] = ...
    random_face_select(centerLat, centerLon)
method = 2; % 1 == original method, 2 == revised method
if method == 1
    % Calculate total length
    totalLength = 0;
    lengthAtNode = zeros(length(centerLat),1);
    lengthAtNode(1) = 0;
    for i=1:length(centerLat)-1
        tmpDeg = distance(centerLat(i), centerLon(i), centerLat(i+1), c...
...enterLon(i+1));
        tmpDist = deg2nm(tmpDeg);
        totalLength = totalLength + tmpDist;
        lengthAtNode(i+1) = lengthAtNode(i) + tmpDist;
    end
    % Convert: 0 to 1
    rouletteWheelAtNode = lengthAtNode/totalLength;
    enterFaceRandom = rand;
    for i=1:length(centerLat)-1
       if(and(rouletteWheelAtNode(i) <= enterFaceRandom, ...
               rouletteWheelAtNode(i+1) >= enterFaceRandom))
          enterFace = i;
          break;
       end
    end
    exitFace = enterFace;
    while(exitFace == enterFace)
       exitFaceRandom = rand;
       for i=1:length(centerLat)-1
          if(and(rouletteWheelAtNode(i) <= exitFaceRandom, ...
               rouletteWheelAtNode(i+1) >= exitFaceRandom))
            exitFace = i;
            break;
          end
       end
    end
    % Get the entering and exiting locations
    enterFaceRand = rand;
    enterFaceLat = centerLat(enterFace) + ...
        enterFaceRand*(centerLat(enterFace+1)-centerLat(enterFace));
    enterFaceLon = centerLon(enterFace) + ...
        enterFaceRand*(centerLon(enterFace+1)-centerLon(enterFace));
```

```
    exitFaceRand = rand;
    exitFaceLat = centerLat(exitFace) + ...
       exitFaceRand*(centerLat(exitFace+1) - centerLat(exitFace));
    exitFaceLon = centerLon(exitFace) + ...
       exitFaceRand*(centerLon(exitFace+1) - centerLon(exitFace));
    enterToExitAzimuth = azimuth(enterFaceLat, enterFaceLon, exitFaceLa...
...t,...
       exitFaceLon);
    if(enterToExitAzimuth > 179) % For FL350 then only azimuths 0 to 17...
...9 valid
       % Switch to reverse direction
       tmpLat = enterFaceLat;
       tmpLon = enterFaceLon;
       enterFaceLat = exitFaceLat;
       enterFaceLon = exitFaceLon;
       exitFaceLat = tmpLat;
       exitFaceLon = tmpLon;
       enterToExitAzimuth = azimuth(enterFaceLat, enterFaceLon, exitFac...
...eLat,...
        exitFaceLon);
    end
else % method == 2

    % Create a box to contain the random flight trajectories
    minRandomLat = min(centerLat) - (max(centerLat) - min(centerLat));
    maxRandomLat = max(centerLat) + (max(centerLat) - min(centerLat));
    lenRandomLat = maxRandomLat - minRandomLat;

    minRandomLon = min(centerLon) - (max(centerLon) - min(centerLon));
    maxRandomLon = max(centerLon) + (max(centerLon) - min(centerLon));
    lenRandomLon = maxRandomLon - minRandomLon;

    enterFaceLat = minRandomLat + rand*lenRandomLat;
    enterFaceLon = minRandomLon + rand*lenRandomLon;

    exitFaceLat = minRandomLat + rand*lenRandomLat;
    exitFaceLon = minRandomLon + rand*lenRandomLon;

    while(true)
       enterInCenter = inpolygon(enterFaceLat,enterFaceLon,centerLat,ce...
...nterLon);
       exitInCenter = inpolygon(exitFaceLat, exitFaceLon, centerLat, ce...
...nterLon);
       [intersLat, intersLon] = polyxpoly([enterFaceLat exitFaceLat], [...
...enterFaceLon exitFaceLon],...
           centerLat, centerLon);
```

```
        if(and(enterInCenter, exitInCenter)) % All points in center
           break;
        elseif(enterInCenter) % Enter point in, exit out
            exitFaceLat = intersLat;
            exitFaceLon = intersLon;
            break;
        elseif(exitInCenter) % Enter point out, exit in
            enterFaceLat = intersLat;
            enterFaceLon = intersLon;
            break;
        elseif(not(isempty(intersLat))) % Both points out, crosses
            enterFaceLat = intersLat(1);
            enterFaceLon = intersLon(1);
            exitFaceLat = intersLat(2);
            exitFaceLon = intersLon(2);
            break;
        end

         enterFaceLat = minRandomLat + rand*lenRandomLat;
         enterFaceLon = minRandomLon + rand*lenRandomLon;

         exitFaceLat = minRandomLat + rand*lenRandomLat;
         exitFaceLon = minRandomLon + rand*lenRandomLon;

    end

end
enterToExitAzimuth = azimuth(enterFaceLat, enterFaceLon, exitFaceLat,.....
....
   exitFaceLon);
if(enterToExitAzimuth > 179) % For FL350 then only azimuths 0 to 179 va...
...lid
   % Switch to reverse direction
   tmpLat = enterFaceLat;
   tmpLon = enterFaceLon;
   enterFaceLat = exitFaceLat;
   enterFaceLon = exitFaceLon;
   exitFaceLat = tmpLat;
   exitFaceLon = tmpLon;
   enterToExitAzimuth = azimuth(enterFaceLat, enterFaceLon, exitFaceLat...
...,...
    exitFaceLon);
end
```

APPENDIX D: DESCRIPTION OF AirPlanJ INPUT AND OUTPUT DATA

This appendix contains a description of the input data files used in the Java program AirPlanJ. The output files are in a format suitable for RAMS input, a standard linear programming format (LP), and the keyhole markup language (kml). A user interested in these file formats should consult the appropriate manuals.

Aircraft Data (acperf.dat)

The following is a description of the data fields:

Data Field	Description
Acperformance	FAA approved aircraft model
Lookup	Aircraft performance lookup group
Description	Long form text description of this aircraft
Cruise Upper	Upper limit of cruising altitude for this flight (100's feet)
Cruise Lower	Lower limit of cruising altitude for this flight (100's feet)
Cruise Optimal	Optimal altitude for cruising for this flight (100's feet)
Range	Maximum stage length distance for this flight (nautical miles)
Not Used	Not Used
Not Used	Not Used
Acgroup	Weight category for this aircraft (e.g. Heavy)

Sample data is shown below:

```
A10 64 "FAIRCHILD THUNDERBOLT II" 500 100 370 9999 future future MEDIUM
A124 18 "ANTONOV ANTONOV AN-124" 370 100 330 9999 future future MEDIUM
A300 02 "AIRBUS A300" 390 150 330 9999 future future HEAVY
```

Preprocessed Climb and Descent Trajectories by Aircraft Group (acperfpreprocess.csv)

The following is a description of the data fields:

Data Field	Description
Acperformance	FAA approved aircraft model
Time	Not used
Flight Phase	Phase of flight (Climb, Cruise, or Descent)
Event	Not used
Not used	Not used
Not used	Not used
Latitude	Latitude location output from preprocessing
Longitude	Longitude location output from preprocessing
Altitude	Altitude output from preprocessing
Time Elapsed	Time elapsed since departure (minutes)

Sample data is shown below:

```
A10,15:02:00,Climb,extNav0,252,4500,38.94453,-77.45581,3.13,0
A10,15:02:00,Climb,IAD,252,4500,38.94453,-77.45581,3.13,0
A10,15:02:02,Climb,#PB,254,4510,38.944409,-77.459547,5,0.17466
```

Preprocessed Aircraft Climb and Descent Distances and Times (aircraftClimbDescent.csv)

The following is a description of the data fields:

Data Field	Description
Acperformance	FAA approved aircraft model
Flight Level	Altitude (100's feet)
Climb Distance	Distance (nm) required to climb to altitude
Climb Time	Time (minutes) required to climb to altitude
Descend Distance	Distance (nm) required to descend from altitude to ground
Descend Time	Time (minutes) required to descend from altitude to ground

Sample data is shown below:

```
SBR2,10.00000,0.66358,0.28333,1.70000,0.73333
SBR2,20.00000,1.64680,0.68333,4.70000,1.83333
SBR2,30.00000,2.66310,1.05000,7.00000,2.45000
```

Airport Information (airports.csv)

The following is a description of the data fields:

Data Field	Description
Airport FAA Id	3-Letter FAA identifier for airport
Description	Short description of the airport
Latitude	Latitude of the airport
Longitude	Longitude of the airport
Elevation	Elevation of the airport (ft)

Sample data is shown below:

```
ADK,ADAK,51.87796389,-176.6460306,18
AKK,AKHIOK,56.93869083,-154.1825556,44
Z13,AKIACHAK,60.90483333,-161.4224444,25
```

Airport Alias and Hub Type (airportinfo.csv)

The following is a description of the data fields:

Data Field	Description
Airport Id1	Either the 3-letter FAA identifier or the 4-letter ICAO identifier
Airport Id2	If the FAA identifier is used for Id1 then this is the ICAO identifier, otherwise it is the FAA identifier
Hub Size	Hub size for this airport

Sample data is shown below:

```
ATL, KATL, Large
BOS, KBOS, Large
KCVG, CVG, Large
```

Domestic Airways Definition (airways.csv)

The following is a description of the data fields:

Data Field	Description
Airway Id	Identifier for this airway, jet route, or RNAV route
Type	Type for this airway (A for Alaska airways)
Sequence	Sequence number for the waypoint
Waypoint Type	If the point is a FIX or a NAVAID
Waypoint Id	Identifier for the waypoint
Latitude	Latitude of the waypoint
Longitude	Longitude of the waypoint

Sample data is shown below:

```
A001,A,10,NAVAID,ZP,53.19611,-131.7772494
A001,A,30,FIX,MOCHA,54.50670833,-133.0209528
A001,A,40,FIX,18436,54.50486111,-133.0306639
```

International Airways Definition (intlairways.csv)

The following is a description of the data fields:

Data Field	Description
Airway Id	Identifier for this airway, jet route, or RNAV route
Direction	Direction (East or West) for this airway
Waypoint1 Id	Identifier of waypoint 1
Waypoint1 County	Two letter country code for waypoint 1
Waypoint1 Latitude	Latitude of waypoint 1
Waypoint1 Longitude	Longitude of waypoint 1
Waypoint2 Id	Identifier of waypoint 2
Waypoint2 County	Two letter country code for waypoint 2
Waypoint2 Latitude	Latitude of waypoint 2
Waypoint2 Longitude	Longitude of waypoint 2

Sample data is shown below:

```
1AW1,E,ORNAT,LY,20.000000,25.000000,DOG,SU,19.182572,30.427283
1AW1,E,DOG,SU,19.182572,30.427283,MRW,SU,18.449772,31.818658
1AW1,E,MRW,SU,18.449772,31.818658,PSD,SU,19.401178,37.241714
```

Airspace Fix (fix.csv, natfix.csv)

The following is a description of the data fields:

Data Field	Description
Fix Id	Fix identifier
Latitude	Latitude for the fix
Longitude	Longitude for the fix
Type	Type of the fix (not used)

Sample data is shown below:

```
10021,31.35066667,-87.52098611,ARTCC-BDRY
10035,34.61783333,-87.43488889,AWY-INTXN
10398,31.51882778,-88.11125556,ARTCC-BDRY
```

Kernel Smoothed Probability Density Functions (kernel_densities.csv)

The following is a description of the data fields:

Data Field	Description
Cluster Type	Type for this cluster (airport, cluster, airport type)
LAT	Look-ahead time (minutes)
Time X	X values of the density as time (minutes)
Density Y	Frequency/density Y values

Sample data is shown below:

```
cluster9,0,4,0.020959653
cluster9,0,5,0.020021758
cluster9,0,6,0.019131857
```

Airspace Navigational Aids - NAVAIDS (navaids.csv)

The following is a description of the data fields:

Data Field	Description
NAVAID ID	NAVAID identifier
Latitude	Latitude of the NAVAID
Longitude	Longitude of the NAVAID

Sample data is shown below:

```
FVE,47.26809583,-68.25670583
FUZ,32.88945,-97.17942528
FTZ,38.69449528,-90.97125111
```

Airspace Sector Definition (sectors.csv)

The following is a description of the data fields:

Data Field	Description
Sector Module	Identifier for the sector module
Sequence Number	Not used
Floor	Floor of the sector module (100's feet)
Ceiling	Ceiling of the sector module (100's feet)
Corners	Latitude/Longitude pairs that define this module

Sample data is shown below:

```
ZSEPD01M1,1,50,130, 45.183333 -122.000000, 45.316667 -121.866667,
   45.366667 -121.833333, 45.450000 -121.783333, 45.466667 -121.933333,
   45.650000 -121.883333, 45.866667 -121.983333, 46.000000 -122.166667,
   46.100000 -122.283333, 46.094444 -122.526389, 46.095833 -122.586111,
   46.000000 -123.000000, 45.850000 -123.233333, 45.700000 -123.316667,
   45.466667 -123.316667, 45.408333 -123.283333, 45.300000 -123.216667,
   45.200000 -122.900000, 45.116667 -122.916667, 45.066667 -122.750000,
   45.066667 -122.466667, 45.127778 -122.308333, 45.166667 -122.183333,
   45.183333 -122.000000
```

Intersection of Airways and Airspace Sectors (airwaySectorIntersection.csv)

The following is a description of the data fields:

Data Field	Description
Airway Id	Identifier for the airway
Waypoint 1	From waypoint for the airway
Waypoint 2	To waypoint for the airway
Center	ARTCC for the intersecting sector
Sector	Sector that intersects with the airway
Not Used	Not Used
Not Used	Not Used
Floor	Floor of the sector module that intersects with the airway (100's feet)
Ceiling	Ceiling of the sector module that intersects with the airway (100's feet)
Latitude	Latitude of the intersection
Longitude	Longitude of the intersection
Elevation	Altitude at the intersection (100's feet)

Sample data is shown below:

```
V258,HVQ,SCRIB,ZID,ZID8601,1,14,231,350,38.0703,-81.4509,290.5
V258,HVQ,SCRIB,ZID,ZID2501,1,12,101,231,38.2521,-81.6581,166
V258,HVQ,SCRIB,ZID,ZID8501,1,4,231,350,38.0703,-81.4509,290.5
```

Location of Fix Within a Sector (fixsector.csv)

The following is a description of the data fields:

Data Field	Description
Fix Id	Fix Identifier
Center	ARTCC that contains the fix
Sector	Sector that contains the fix
Floor	Floor (100's feet) of the sector module that contains the fix
Ceiling	Ceiling (100's feet) of the sector module that contains the fix

Sample data is shown below:

```
BUKKO,ZAB,ZAB7993,370,1000
13196,ZAB,ZAB7993,370,1000
EKIYO,ZAB,ZAB7993,370,1000
```

Location of NAVAID Within a Sector (navaidsector.csv)

The following is a description of the data fields:

Data Field	Description
NAVAID Id	NAVAID Identifier
Center	ARTCC that contains the NAVAID
Sector	Sector that contains the NAVAID
Floor	Floor (100's feet) of the sector module that contains the NAVAID
Ceiling	Ceiling (100's feet) of the sector module that contains the NAVAID

Sample data is shown below:

```
P14,ZAB,ZAB7993,370,1000
GNT,ZAB,ZAB7993,370,1000
RQE,ZAB,ZAB7993,370,1000
```

Sector Area and Volume (sector_area.csv)

The following is a description of the data fields:

Data Field	Description
Sector	Sector identifier
Area	Plan area of the sector (nm^2)
Volume	Volume of airspace in the sector (nm^3)

Sample data is shown below:

```
ZAB7993, 9047.905916, 93813.097216
ZAB4244, 14296.508942, 31331.191188
ZAB4301, 14605.160061, 47268.743167
```

Sector Monitor Alert Parameter (MAP) Value (sectormap.csv)

The following is a description of the data fields:

Data Field	Description
Sector	Sector identifier
MAP	Monitor Alert Parameter value

Sample data is shown below:

```
ZAU2201,8
ZAU2324,12
ZAU2325,12
```

Standard Terminal ARrival (STAR) and Departure Path (DP) (stardp.csv)

The following is a description of the data fields:

Data Field	Description
Latitude	Latitude of the STAR or DP
Longitude	Longitude of the STAR or DP
Fix	Fix or NAVAID of the STAR or DP
STAR/DP	Identifier for STAR or DP

Sample data is shown below:

```
32.48111111,-99.86333333,ABI,ABI.JEN8
32.48111111,-99.86333333,ABI,ABI.KNEAD5
32.48111111,-99.86333333,ABI,ABI.SLUGG5
```

Weibull Parameters to Generate Random Convective Weather Forecasts (ncwf_weibull_parameters.csv)

The following is a description of the data fields:

Data Field	Description
Forecast Area	Area of the convective weather forecast (nm^2)
Lambda Bias 0	Weibull Lambda parameter for a 0 nm buffer
K Bias 0	Weibull K parameter for a 0 nm buffer
Lambda Bias 10	Weibull Lambda parameter for a 10 nm buffer
K Bias 10	Weibull K parameter for a 10 nm buffer
Lambda Bias 20	Weibull Lambda parameter for a 20 nm buffer
K Bias 20	Weibull K parameter for a 20 nm buffer
Lambda Centroid	Weibull Lambda parameter for centroid-to-centroid distance
K Centroid	Weibull K parameter for centroid-to-centroid distance

Sample data is shown below:

```
0,1.35389,0.82621,9.91347,0.87981,26.06934,0.86799,81.75977,0.47189
50,1.74525,0.83171,11.35276,0.86008,29.08621,0.84088,75.03256,0.50731
100,3.17579,0.89381,16.06052,0.81037,39.02360,0.77019,61.50685,0.87242
```

Collaborative Convective Forecast Product (CCFP) Echo Tops Forecast (convection_tops.csv)

The following is a description of the data fields:

Data Field	Description
Hour	Hour of forecast
Group	Group number for the forecast
Latitude	Latitude of the group centroid
Longitude	Longitude of the group centroid
Forecasted Tops	Forecasted top for the convection group
Observed Tops	Observed top for the convection group

Sample data is shown below:

```
21,1,37.379,-79.566,3,2
21,2,34.433,-83.96,3,2
21,3,47.232,-93.56,2,1
```

Closest Wind Station for Fixes and NAVAIDs (fixStation.csv, natfixStation.csv, navaidStation.csv)

The following is a description of the data fields:

Data Field	Description
Airspace Id	Fix or NAVAID identifier
Weather Station	Weather station identifier

Sample data is shown below:

```
ZZAPP,JAN
ZYDCO,MSY
ZUZXE,AMA
```

Wind Station Locations (station_locations.csv)

The following is a description of the data fields:

Data Field	Description
Weather Station	Weather station identifier
Latitude	Latitude of the weather station
Longitude	Longitude of the weather station

Sample data is shown below:

```
FSM,35.38333333333333,-94.26666666666667
IMB,44.63333333333333,-119.7
BTI,70.16666666666667,-143.91666666666666
```

Wind Forecast (0ZWindData.csv, 600ZWindData.csv, 1200ZWindData.csv, 1800ZWindData.csv)

The following is a description of the data fields:

Data Field	Description
Latitude	Latitude of the weather station
Longitude	Longitude of the weather station
Altitude	Altitude of the wind forecast (100's feet)
Direction	Direction of the wind forecast
Magnitude	Magnitude of the wind forecast (knots)

Sample data is shown below:

```
30.683333333333334,-88.23333333333333,90,110,26
30.683333333333334,-88.23333333333333,120,110,27
30.683333333333334,-88.23333333333333,180,130,24
```

APPENDIX E: AirPlanJ SOURCE CODE

This appendix contains the source code for the Java program AirPlanJ. AirPlanJ.java is the main file and entry point to the program.

AirPlanJ.java

```
//
//  AirPlanJ.java
//  AirPlanJ
//
//  Created by jeff on 7/24/06.
//  Copyright (c) 2006. All rights reserved.
//
import java.util.*;
import java.io.*;
public class AirPlanJ {

    /**
     *
     * @param args
     */
    public AirPlanJ(){

    }

    public static void main(String args[]) {
        /** For heap memory errors use the "-Xmx1024m" option */
        /** Timer for system performance */
        long startTime = System.currentTimeMillis();


        initializationToAssignment();

        kShortestPathsToKml();

        /** Print the program execution time */
        System.out.print("Program Execution Time: ");
        double elapsedTime = (System.currentTimeMillis() - startTime)/1...
...000D;
        System.out.println(elapsedTime);
    }

    public static double[][] doubleArrayCopy(double[][] src){
        double[][] dest = new double[src.length][src[0].length];

        for(int i=0; i<src.length; i++){
            for(int j=0; j<src[i].length; j++){
                dest[i][j] = src[i][j];
            }
        }

        return dest;
```

```
    }

    // Initialization of data to assignment of flights to planned netwo...
...rk
    public static void initializationToAssignment(){
        /** Initialize static global objects */
        APData.Initialize();

        Settings.defaultSettings(APData.airspace);

        APData.InitializeMatlab();

        /** Read databases and ETMS data */
        FAPio.readData();

        /** Read convection forecast */
        FAPioWeatherReader wr = new FAPioWeatherReader();
        wr.convectionForecast();

        /** Create the network and kml file */
        NetworkBuilder.buildNetwork();
        FAPioNetworkReader nr = new FAPioNetworkReader(APData.network);...
...

        /** Calculate length of links */
        try{
            nr.readLinkLengths();
        }catch(IOException e){
            System.out.println("Error reading network links.");
        }
        APData.network.calculateLinkLengths();

        /** Get time and distance required for climb and descent */
        FAPioAircraftReader ar = new FAPioAircraftReader();
        try{
            ar.readClimbDescentDistanceTime(Settings.aircraftClimbDesce...
...ntFile);
        }catch(IOException e){
            System.out.println("Error reading climb/descent distances."...
...);
        }

        /** Assign planned flights to the network */
        System.out.println("Assigning planned flights to the network.")...
...;
        APData.network.assignPlannedFlights();
        /** Must read the weather data after reading the sector info */...
```

```
...        wr.scenarioCapacityCalculator();

        /** Get constrained sectors */
        System.out.println("Number of constrained sectors: " +
                Integer.toString(
                APData.network.calculateFlowConstrainedSectors().size()...
...));

        /** Cluster flights */
        Scenario fpScenario = APData.network.getFlightPlanScenario();
        fpScenario.clusterFlights();

        /** Terminate the library */
        APData.TerminateMatlab();


    }

    public static void kShortestPathsToKml(){
        final double penaltyAltitudeOutOfRange = 1.2D;
        final int numKPaths = 10;

        // k-Shortest path calculations
        Scenario flightPlanScenario = APData.network.getFlightPlanScena...
...rio();
        flightPlanScenario.kShortestPathCalculations(numKPaths,
                penaltyAltitudeOutOfRange);
        // Create the Integer Programming APCDM-CPLEX input file
        FAPioApcdmInputWriter aiw = new FAPioApcdmInputWriter(flightPla...
...nScenario);

        // Incremental assignment calculations
        APData.network.incrementalAssignment(penaltyAltitudeOutOfRange)...
...;

        // Write the results for verification
        FAPioKmlWriter kw = new FAPioKmlWriter();
        kw.writeKmlFile();

        // Write a log file for debugging purposes
        FAPioLogWriter lw = new FAPioLogWriter();

        // Write additional RAMS input files for simulation validation
        FAPio.writeRamsData();
        FAPioRamsWriter rw = new FAPioRamsWriter(APData.airspace);
        rw.writeWeatherRestriction();
        rw.writeThresholdCapacity();
```

```
            rw.writeWeatherTrafficAndTrafficProfile(numKPaths);
        }


    }
```

ACenter.java

```
//
//  ACenter.java
//  AirPlanJ
//
//  Created on 8/2/06.
//  Copyright 2006. All rights reserved.
//
/**
 *
 * @author jeff
 */
import java.util.*;
public class ACenter {
    static final int INIT_NUM_SECTORS = 200;

    private String name_;
    private HashMap <String,ASector> sectors_;

    private int[] hourlyDemand_;

    private double minLatitude_;
    private double maxLatitude_;
    private double minLongitude_;
    private double maxLongitude_;

        /* Creates a new instance of an Airspace Center object
         *  with initial capacity for 200 sectors.
         */
    public ACenter(String name){
        name_ = name;
        sectors_ = new HashMap <String,ASector>(INIT_NUM_SECTORS, Setti...
...ngs.hashMapLoadFactor);
        hourlyDemand_ = new int[25];

        minLatitude_ = Double.MAX_VALUE;
        maxLatitude_ = Double.MIN_VALUE;
        minLongitude_ = Double.MAX_VALUE;
        maxLongitude_ = Double.MIN_VALUE;
    }

    private ACenter calculateMinMaxLatLon(){
        for(ASector sector:sectors_.values()){
            double minLat = sector.getMinimumLat();
            double maxLat = sector.getMaximumLat();
            double minLon = sector.getMinimumLon();
            double maxLon = sector.getMaximumLon();
```

```
        if(minLat < minLatitude_) minLatitude_ = minLat;
        if(maxLat > maxLatitude_) maxLatitude_ = maxLat;
        if(minLon < minLongitude_) minLongitude_ = minLon;
        if(maxLon > maxLongitude_) maxLongitude_ = maxLon;
    }
    return this;
}

public double getMinimumLat(){
    if(minLatitude_ == Double.MAX_VALUE) calculateMinMaxLatLon();

    return minLatitude_;
}

public double getMaximumLat(){
    if(maxLatitude_ == Double.MIN_VALUE) calculateMinMaxLatLon();

    return maxLatitude_;
}

public double getMinimumLon(){
    if(minLongitude_ == Double.MAX_VALUE) calculateMinMaxLatLon();

    return minLongitude_;
}

public double getMaximumLon(){
    if(maxLongitude_ == Double.MIN_VALUE) calculateMinMaxLatLon();

    return maxLongitude_;
}

/** Get center name */
public String getName(){
    return name_;
}

/** Add a sector (no longer used) */
public ACenter addSector(ASector sec){
    sectors_.put(sec.getName(),sec);
    return this;
}

/** Add a sector module and create a sector if necessary */
public ACenter addSectorModule(ASectorModule secMod){
    // Get the sector name
```

```
        String sectorModuleName = secMod.getName();
        int lastM = sectorModuleName.lastIndexOf("M");
        lastM = lastM == -1 ? sectorModuleName.lastIndexOf(APData.X) : ...
...lastM;
        String sectorName = sectorModuleName.substring(0, lastM);

        // Get reference to sector
        ASector s;
        if(sectors_.containsKey(sectorName)){
            s = sectors_.get(sectorName);
        }
        else{
            s = new ASector(sectorName);
        }
        sectors_.put(sectorName, s);
        s.addModule(secMod);
        return this;
    }

    /** Number of sectors. Used to iterate through sector list
     * using getSector method */
    public int noSectors(){
        return sectors_.size();
    }

    /** Iterate through sectors */
    public Iterator<ASector> getSectorValueIterator(){
        return sectors_.values().iterator();
    }

    /** Get a sector by index */
    public ASector getSector(String name){
        return sectors_.get(name);
    }

    public boolean containsSector(String name){
        return sectors_.containsKey(name);
    }

    public ACenter addHourlyDemand(int hour){
        hourlyDemand_[hour]++;
        return this;
    }

    public double getHourlyDemand(int hour){
        // Divide by two since hourly demand includes both enter and ex...
...it demand
```

```
        return hourlyDemand_[hour]/2.0D;
    }

}
```

Airport.java

```
/*
 * Airport.java
 *
 * Created on August 3, 2006, 10:13 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
/**
 *
 * @author Jeff Henderson
 */
import java.util.*;
public class Airport extends Node3D<Double>{
    /** FAA airport identifier */
    private String identifier_;

    /** FAA US international airport identifier */
    private String kIdentifier_;

    /** Description (name) of airport */
    private String description_;

    /** Type of airport (large hub, medium hub, ...) */
    private String airportType_;

    /** Creates a new instance of Airport without ICAO "K" identifier *...
.../
    public Airport(String identifier, GPoint3D<Double> location) {
        super(location, Node3D.NodeTypes.Airport, identifier);
        identifier_ = identifier;
        kIdentifier_ = APData.NULLSTRING;
        description_ = APData.NULLSTRING;
        airportType_ = APData.NULLSTRING;
    }

    /** Airport object with optional description */
    public Airport(String identifier, String description, GPoint3D<Doub...
...le> location){
        super(location, Node3D.NodeTypes.Airport, identifier);
        identifier_ = identifier;
        kIdentifier_ = APData.NULLSTRING;
        description_ = description;
        airportType_ = APData.NULLSTRING;
    }
```

```
    /** Creates a new instance of Airport with ICAO "K" initialized */
    /*public Airport(String identifier, String kIdentifier, GPoint3D<Do...
...uble> location){
        identifier_ = identifier;
        kIdentifier_ = kIdentifier;
        location_ = location;
    }*/

    /** Returns identifier (without "K" prefix for U.S. airports) */
    public String getIdentifier(){
        return identifier_;
    }

    /** Adds ICAO "K" identifier if 3-Letter FAA code created first */
    public Airport setIcaoKIdentifier(String kIdentifier){
        kIdentifier_ = kIdentifier;
        return this;
    }

    /** 4-Letter identifier for US airports beginning with "K" */
    public String getKIdentifier(){
        return kIdentifier_;
    }

    /** Long description of airport */
    public String getDescription(){
        return description_;
    }

    public Airport setAirportType(String airportType){

        airportType_ = airportType.replace("-","");
        return this;
    }

    public String getAirportType(){
        //Use Non-Hub as default
        if(airportType_.equals(APData.NULLSTRING)) return "NonHub";
        return airportType_;
    }

}
```

Airspace.java

```
//
//  Airspace.java
//  AirPlanJ
//
//  Created on 8/2/06.
//
//
/**
 *
 * @author jeff
 */
import java.util.*;
public class Airspace {
    /* Number of centers in airspace should be constant */
    static final int INIT_NUM_CENTERS = 22;

    /* Pre-allocate based on known number of airports */
    static final int INIT_NUM_AIRPORTS = 20000;

    /* Pre-allocate number of aircraft based on daily maximum in ETMS *...
.../
    static final int INIT_NUM_FLIGHTS = 100000;

    /* Pre-allocate number of navaids based on FAA database */
    static final int INIT_NUM_NAVAIDS = 2500;

    /* Pre-allocate number of fixes based on FAA database */
    static final int INIT_NUM_FIXES = 40000;

    /* Pre-allocate number of airways based on FAA database */
    public static final int INIT_NUM_AIRWAYS = 1000;

    /* Number of international airways */
    public static final int INIT_NUM_INTL_AIRWAYS = 7500;

    /* Number of wind stations (fixed) */
    static final int NUM_WIND_STATIONS = 227;

    /** Key/Value storage of centers based on 3-Letter identifiers */
    private HashMap<String, ACenter> centers_;

    /** Key/Value storage of airports based on 3-Letter identifiers */
    private HashMap<String, Airport> airports_;

    /** Key/Value storage of flights based on callsign */
    private HashMap<String, Flight> flights_;
```

```
    /** List of flights to exclude (filter) based on departure or arriv...
...al time */
    private ArrayList<String> flightFilter_;

    /** List of actual departure times */
    private HashMap<String, String> flightDepartureTimes_;

    /** Key/Value storage of NAVAIDs based on identifier */
    private HashMap<String, GPoint2D<Double>> navaids_;

    /** Key/Value storage of fixes and fix types */
    private HashMap<String, GPoint2D<Double>> fixes_;
    private HashMap<String, String> fixTypes_;
    private HashMap<String, ArrayList<String>> duplicateFixNavaidIdenti...
...fiers_;

    /** Key/Value storage of standard terminal arrivals (STARs) */
    private HashMap<String, GPoint2D<Double>> stars_;
    private HashMap<String, String> starAirspaceIdentifiers_;

    /** Key/Value storage of VOR (victor) airways */
    private HashMap<String, Airway> victorAirways_;

    /** Key/Value storage of jet routes */
    private HashMap<String, Airway> jetRoutes_;

    /** Key/Value storage of RNAV routes */
    private HashMap<String, Airway> rnavRoutes_;

    /** For each airspace navaid/fix list airways using it */
    private HashMap<String, ArrayList<Airway>> victorAirwaysByIdentifie...
...r_;
    private HashMap<String, ArrayList<Airway>> jetRoutesByIdentifier_;
    private HashMap<String, ArrayList<Airway>> rnavRoutesByIdentifier_;...
...
    /** Preferred routes between airport pairs */
    private HashMap<String, HashMap<String, String>> preferredRoutes_;

    /** Filter for flight tracks based on location */
    private java.awt.Polygon locationFilter_;

    /** Data structure to hold flight level statistical data */
    private HashMap<Integer, ArrayList<Double>> flightLevelStats_;

    /** 3-Letter computer identifier (CID) which may (or may not) be un...
...ique to each flight */
```

```
    private HashMap<String, ArrayList<String>> flightComputerIdentifier...
...s_;

    /** Store the number of sectors for array creation */
    private int noSectors_;

    public Airspace(){
        centers_ = new HashMap<String, ACenter>(INIT_NUM_CENTERS, Setti...
...ngs.hashMapLoadFactor);
        airports_ = new HashMap<String, Airport>(INIT_NUM_AIRPORTS, Set...
...tings.hashMapLoadFactor);
        flights_ = new HashMap<String, Flight>(INIT_NUM_FLIGHTS, Settin...
...gs.hashMapLoadFactor);
        flightDepartureTimes_ = new HashMap<String, String>(INIT_NUM_FL...
...IGHTS, Settings.hashMapLoadFactor);
        navaids_ = new HashMap<String, GPoint2D<Double>>(INIT_NUM_NAVAI...
...DS, Settings.hashMapLoadFactor);
        fixes_ = new HashMap<String, GPoint2D<Double>>(INIT_NUM_FIXES, ...
...Settings.hashMapLoadFactor);
        fixTypes_ = new HashMap<String, String>(INIT_NUM_FIXES, Setting...
...s.hashMapLoadFactor);
        duplicateFixNavaidIdentifiers_ = new HashMap<String, ArrayList<...
...String>>(INIT_NUM_FIXES, Settings.hashMapLoadFactor);
        stars_ = new HashMap<String, GPoint2D<Double>>(Settings.hashMap...
...DefaultInitialCapacity,
                Settings.hashMapLoadFactor);
        starAirspaceIdentifiers_ = new HashMap<String, String>(Settings...
....hashMapDefaultInitialCapacity,
                Settings.hashMapLoadFactor);

        victorAirways_ = new HashMap<String, Airway>(INIT_NUM_AIRWAYS, ...
...Settings.hashMapLoadFactor);
        jetRoutes_ = new HashMap<String, Airway>(INIT_NUM_AIRWAYS + INI...
...T_NUM_INTL_AIRWAYS, Settings.hashMapLoadFactor);
        rnavRoutes_ = new HashMap<String, Airway>(INIT_NUM_AIRWAYS, Set...
...tings.hashMapLoadFactor);

        victorAirwaysByIdentifier_ = new HashMap<String, ArrayList<Airw...
...ay>>(
                INIT_NUM_AIRWAYS, Settings.hashMapLoadFactor);
        jetRoutesByIdentifier_ = new HashMap<String, ArrayList<Airway>>...
...(
                INIT_NUM_AIRWAYS, Settings.hashMapLoadFactor);
        rnavRoutesByIdentifier_ = new HashMap<String, ArrayList<Airway>...
...>(
                INIT_NUM_AIRWAYS, Settings.hashMapLoadFactor);
```

```
        preferredRoutes_ = new HashMap<String, HashMap<String, String>>...
...(
                Settings.hashMapDefaultInitialCapacity, Settings.hashMa...
...pLoadFactor);
        flightLevelStats_ = new HashMap<Integer, ArrayList<Double>>(INI...
...T_NUM_FLIGHTS, Settings.hashMapLoadFactor);
        flightComputerIdentifiers_ = new HashMap<String, ArrayList<Stri...
...ng>>(INIT_NUM_FLIGHTS, Settings.hashMapLoadFactor);
        locationFilter_ = new java.awt.Polygon(Settings.filterLongitude...
..., Settings.filterLatitude, Settings.filterLongitude.length);

        flightFilter_ = new ArrayList<String>();
    }

    public Airspace addVictorAirwayToIdentifier(String id, Airway awy){...
...        ArrayList<Airway> awys = victorAirwaysByIdentifier_.containsKey...
...(id) ?
            victorAirwaysByIdentifier_.get(id) : new ArrayList<Airway>(...
...);

        if(!awys.contains(awy)) awys.add(awy);

        victorAirwaysByIdentifier_.put(id, awys);

        return this;
    }

    public Airspace addJetRouteToIdentifier(String id, Airway awy){
        ArrayList<Airway> awys = jetRoutesByIdentifier_.containsKey(id)...
... ?
            jetRoutesByIdentifier_.get(id) : new ArrayList<Airway>();

        if(!awys.contains(awy)) awys.add(awy);

        jetRoutesByIdentifier_.put(id, awys);


        return this;
    }

    public Airspace addRnavRouteToIdentifier(String id, Airway awy){
        ArrayList<Airway> awys = rnavRoutesByIdentifier_.containsKey(id...
...) ?
            rnavRoutesByIdentifier_.get(id) : new ArrayList<Airway>();

        if(!awys.contains(awy)) awys.add(awy);
```

```
        rnavRoutesByIdentifier_.put(id, awys);


        return this;
    }

    public boolean onVictorAirway(String id){
        return victorAirwaysByIdentifier_.containsKey(id);
    }

    public boolean onJetRoute(String id){
        return jetRoutesByIdentifier_.containsKey(id);
    }

    public boolean onRnavRoute(String id){
        return rnavRoutesByIdentifier_.containsKey(id);
    }

    public ArrayList<Airway> victorAirwaysByIdentifier(String id){
        return victorAirwaysByIdentifier_.get(id);
    }

    public ArrayList<Airway> jetRoutesByIdentifier(String id){
        return jetRoutesByIdentifier_.get(id);
    }

    public ArrayList<Airway> rnavRoutesByIdentifier(String id){
        return rnavRoutesByIdentifier_.get(id);
    }

        /* Add an airway point with an identifier and create the airway...
...         * if it doesn't exist
         */
    public Airspace addAirwayPoint(GPoint2D<Double> pt, String ptId, St...
...ring airwayDesignator){
        Airway a;

        boolean isJetRoute = false;
        boolean isRnavRoute = false;
        boolean isVictorAirway = false;

        HashMap<String, Airway> ac = null;

        if(airwayDesignator.startsWith(APData.J)){
            /* Jet routes start with 'J' */
            isJetRoute = true;
            ac = jetRoutes_;
```

```
        } else if(airwayDesignator.startsWith(APData.Q)){
            /* GPS RNAV Routes start with 'Q' */
            isRnavRoute = true;
            ac = rnavRoutes_;
        } else if(airwayDesignator.startsWith(APData.V) || airwayDesign...
...ator.startsWith(APData.A)
        || airwayDesignator.startsWith(APData.B) || airwayDesignator.st...
...artsWith(APData.G)
        || airwayDesignator.startsWith(APData.R)){
            isVictorAirway = true;
            ac = victorAirways_;
        }else{
            /* Default airway type is jet routes */
            isJetRoute = true;
            ac = jetRoutes_;
        }

        if(ac.containsKey(airwayDesignator)){
            a = ac.get(airwayDesignator);

            // Don't add duplicate points
            if(!a.onAirway(ptId).equals(APData.FALSE)){
                return this;
            }
        } else{
            a = new Airway(airwayDesignator);
            ac.put(airwayDesignator, a);
        }
        a.addPoint(pt, ptId);

        if(isJetRoute){
            this.addJetRouteToIdentifier(ptId, a);
        }else if(isVictorAirway){
            this.addVictorAirwayToIdentifier(ptId, a);
        }else{
            this.addRnavRouteToIdentifier(ptId, a);
        }

        return this;
    }

    public Airspace addAirwayPoint(GPoint2D<Double> pt, String ptId, St...
...ring airwayDesignator,
            boolean addAfter, String previousPoint){
        Airway a;

        boolean isJetRoute = false;
```

```
        boolean isRnavRoute = false;
        boolean isVictorAirway = false;

        HashMap<String, Airway> ac = null;

        if(airwayDesignator.startsWith(APData.J)){
            /* Jet routes start with 'J' */
            isJetRoute = true;
            ac = jetRoutes_;
        } else if(airwayDesignator.startsWith(APData.Q)){
            /* GPS RNAV Routes start with 'Q' */
            isRnavRoute = true;
            ac = rnavRoutes_;
        } else if(airwayDesignator.startsWith(APData.V) || airwayDesign...
...ator.startsWith(APData.A)
        || airwayDesignator.startsWith(APData.B) || airwayDesignator.st...
...artsWith(APData.G)
        || airwayDesignator.startsWith(APData.R)){
            isVictorAirway = true;
            ac = victorAirways_;
        }else{
            /* Default airway type is jet routes */
            isJetRoute = true;
            ac = jetRoutes_;
        }

        if(ac.containsKey(airwayDesignator)){
            a = ac.get(airwayDesignator);

            // Don't add duplicate points
            if(!a.onAirway(ptId).equals(APData.FALSE)){
                return this;
            }
        } else{
            a = new Airway(airwayDesignator);
            ac.put(airwayDesignator, a);
        }


        if(addAfter){
           a.addPointAfter(pt, ptId, previousPoint);
        }else{
           a.addPointBefore(pt, ptId, previousPoint);
        }

        if(isJetRoute){
            this.addJetRouteToIdentifier(ptId, a);
```

```
    }else if(isVictorAirway){
        this.addVictorAirwayToIdentifier(ptId, a);
    }else{
        this.addRnavRouteToIdentifier(ptId, a);
    }

    return this;
}



/** Get an airway object using the airway designator */
public Airway getAirway(String airwayDesignator){
    if(victorAirways_.containsKey(airwayDesignator)){
        return victorAirways_.get(airwayDesignator);
    }else if(jetRoutes_.containsKey(airwayDesignator)){
        return jetRoutes_.get(airwayDesignator);
    }else{  //rnav route
        return rnavRoutes_.get(airwayDesignator);
    }
}
public Airway getVictorAirway(String airwayDesignator){
    return victorAirways_.get(airwayDesignator);
}
public Airway getJetRoute(String airwayDesignator){
    return jetRoutes_.get(airwayDesignator);
}
public Airway getRnavRoute(String airwayDesignator){
    return rnavRoutes_.get(airwayDesignator);
}

/** Get the airway iterator through the hashmap */
public Iterator<String> getVictorAirwayIterator(){
    return victorAirways_.keySet().iterator();
}
public Iterator<String> getJetRouteIterator(){
    return jetRoutes_.keySet().iterator();
}
public Iterator<String> getRnavRouteIterator(){
    return rnavRoutes_.keySet().iterator();
}

public boolean hasAirway(String airwayId){
    return jetRoutes_.containsKey(airwayId) ||
            rnavRoutes_.containsKey(airwayId) ||
            victorAirways_.containsKey(airwayId);
}
```

```
    public boolean hasJetRoute(String jetRouteId){
        return jetRoutes_.containsKey(jetRouteId);
    }
    public boolean hasRnavRoute(String rnavRouteId){
        return rnavRoutes_.containsKey(rnavRouteId);
    }
    public boolean hasVictorAirway(String victorAirwayId){
        return victorAirways_.containsKey(victorAirwayId);
    }

    /* Add a STAR using identifier and associated NAVAID/FIX */
    public Airspace addStar(String starId, String navaid){
        GPoint2D<Double> starLocation;
        if(navaids_.containsKey(navaid)){
            starLocation = navaids_.get(navaid);
        }else if(fixes_.containsKey(navaid)){
            starLocation = fixes_.get(navaid);
        }else{
            starLocation = new GPoint2D<Double>();
        }

        stars_.put(starId, starLocation);

        return this;
    }

    public Airspace addStar(String starId, String locationId, GPoint2D<...
...Double> location){
        stars_.put(starId, location);
        starAirspaceIdentifiers_.put(starId, locationId);
        return this;
    }

    /* Check to see if there exists a STAR with name */
    public boolean hasStar(String starId){
        return stars_.containsKey(starId);
    }

    /* Get STAR based on identifier */
    public GPoint2D<Double> getStar(String starId){
        return stars_.get(starId);
    }

    public String getStarAirspaceIdentifier(String starId){
        return starAirspaceIdentifiers_.get(starId);
    }
```

```
    /* Get STAR keys to navigate through hashmap */
    public Iterator<String> getStarIterator(){
        return stars_.keySet().iterator();
    }

    /* Add a navigation fix using identifier as a key */
    public Airspace addFix(GPoint2D<Double> fix, String fixId){
        fixes_.put(fixId, fix);

        this.addDuplicateFixNavaidIdentifier(fixId);

        return this;
    }

    private Airspace addDuplicateFixNavaidIdentifier(String identifier)...
...{
        // Add duplicate ids
        //String[] id_ = identifier.split("_");
        //String idBeforeUnderscore = id_[0];
        String idBeforeUnderscore = identifier.contains(APData.UNDERSCO...
...RE) ?
            identifier.substring(0, identifier.indexOf(APData.UNDERSCOR...
...E)) :
            identifier;

        ArrayList<String> duplicateIds = duplicateFixNavaidIdentifiers_...
....containsKey(idBeforeUnderscore) ?
            duplicateFixNavaidIdentifiers_.get(idBeforeUnderscore) : ne...
...w ArrayList<String>();

        if(!duplicateIds.contains(identifier)) duplicateIds.add(identif...
...ier);

        duplicateFixNavaidIdentifiers_.put(idBeforeUnderscore, duplicat...
...eIds);


        return this;
    }

    /* Check to see if data structure contains named fix */
    public boolean hasFix(String fixId){
        return fixes_.containsKey(fixId);
    }

    /* Get a navigational fix based on identifier */
    public GPoint2D<Double> getFix(String fixId){
```

```
        return fixes_.get(fixId);
    }

    public String getClosestFixNavaid(String identifier, GPoint2D<Doubl...
...e> ptToCompare){
        // Initialize with domestic fix identifier
        String idToReturn = identifier;
        GPoint2D<Double> ptToReturn = fixes_.containsKey(identifier) ?
            fixes_.get(identifier) : navaids_.get(identifier);

        Mapping m = new Mapping(ptToReturn.getLatitude(), ptToReturn.ge...
...tLongitude(),
                ptToCompare.getLatitude(),  ptToCompare.getLongitude())...
...;
        double closestDistance = m.getDistance();

        //double closestDistance = distance(ptToReturn, ptToCompare);

        // List of all domestic and international fixes using identifie...
...r
        //String identifier_ = identifier.split("_")[0];
        String identifier_ = identifier.contains(APData.UNDERSCORE) ?
            identifier.substring(0, identifier.indexOf(APData.UNDERSCOR...
...E)) :
            identifier;
        ArrayList<String> duplicateIds = duplicateFixNavaidIdentifiers_...
....get(identifier_);

        for(String candidateId:duplicateIds){
            GPoint2D<Double> candidatePt = fixes_.containsKey(candidate...
...Id) ?
                fixes_.get(candidateId) : navaids_.get(candidateId);

            Mapping m2 = new Mapping(candidatePt.getLatitude(),
                    candidatePt.getLongitude(), ptToCompare.getLatitude...
...(),
                    ptToCompare.getLongitude());
            double candidateDistance = m2.getDistance();

            //double candidateDistance = distance(candidatePt, ptToComp...
...are);
            if(candidateDistance < closestDistance){
                closestDistance = candidateDistance;
                ptToReturn = candidatePt;
                idToReturn = candidateId;
            }
        }
```

```
        return idToReturn;
    }

    /* Get fix keys to iterate through hashmap */
    public Iterator<String> getFixIterator(){
        return fixes_.keySet().iterator();
    }

    /* Add navigation fix type using identifier as a key */
    public Airspace addFixType(String type, String fixId){
        fixTypes_.put(fixId, type);
        return this;
    }

    /* Get navigation fix type using identifier as a key */
    public String getFixType(String fixId){
        return fixTypes_.get(fixId);
    }

    /* Get number of fixes */
    public int noFixes(){
        return fixes_.size();
    }

    /* Add a NAVAID using identifier as a key */
    public Airspace addNavaid(GPoint2D<Double> navaid, String navaidId)...
...{
        navaids_.put(navaidId, navaid);

        this.addDuplicateFixNavaidIdentifier(navaidId);
        return this;
    }

    /* Check to see if data structure contains named NAVAID */
    public boolean hasNavaid(String navaidId){
        return navaids_.containsKey(navaidId);
    }

    /* Get a navaid based on the identifier */
    public GPoint2D<Double> getNavaid(String navaidId){
        return navaids_.get(navaidId);
    }

    /* Get NAVAID keys to iterate through hashmap */
    public Iterator<String> getNavaidIterator(){
        return navaids_.keySet().iterator();
```

```
}

/* Get number of NAVAIDs */
public int noNavaids(){
    return navaids_.size();
}

/* Get center object with 3-letter name key */
public ACenter getCenter(String s){
    return centers_.get(s);
}

/* Get center keys to enumerate through hashmap */
public Iterator<String> getCenterKeyIterator(){
    return centers_.keySet().iterator();
}

/* Get center values to enumerate through hashmap */
public Iterator<ACenter> getCenterValueIterator(){
    return centers_.values().iterator();
}

/* Add a sector and a center if not available */
public Airspace addSectorModule(ASectorModule s){
    String n = s.getName().substring(0,3);

    if(!centers_.containsKey(n)){
        ACenter c = new ACenter(n);
        centers_.put(n, c);
    }

    ACenter c = centers_.get(n);
    c.addSectorModule(s);

    return this;
}

/* Get airport object using identifier */
public Airport getAirport(String s){
    return airports_.get(s);
}

/* Add an airport */
public Airspace addAirport(Airport a){
    airports_.put(a.getIdentifier(), a);
    return this;
}
```

```
/* Add an airport using the airport identifier */
public Airspace addAirport(String identifier, Airport a){
    airports_.put(identifier, a);
    return this;
}

/* Check if airport already in airport list */
public boolean airportExists(String identifier){
    return airports_.containsKey(identifier);
}

/* Get airport keys to enumerate through hashmap */
public Iterator<String> getAirportKeyIterator(){
    return airports_.keySet().iterator();
}

/* Get airport elements to enumerate through hashmap */
public Iterator<Airport> getAirportValueIterator(){
    return airports_.values().iterator();
}

/**  */
public int noAirports(){
    return airports_.size();
}

/* Add a flight */
public Airspace addFlight(Flight f){
    flights_.put(f.getCallSign(), f);
    return this;
}

/** Get the number of stored flights */
public int noFlights(){
    return flights_.values().size();
}

/* Get iterator for flights */
public Iterator<String> getFlightIterator(){
    return flights_.keySet().iterator();
}

/* Retrieves flight object by callsign */
public Flight getFlight(String callSign){
    return flights_.get(callSign);
}
```

```
    /* Checks to see if flight already in airspace object */
    public boolean containsFlight(String callSign){
        return flights_.containsKey(callSign);
    }

    /* Checks to see if flight track is in the filter location */
    public boolean flightInRange(double longitude, double latitude, dou...
...ble altitude){
        return locationFilter_.contains(longitude, latitude) &&
                altitude > Settings.filterMinAltitude &&
                altitude < Settings.filterMaxAltitude;
    }
    public boolean flightInRange(double longitude, double latitude){
        return (longitude > -105 && longitude < -63 && latitude > 25
                && latitude < 47);

        //return locationFilter_.contains(longitude, latitude);
    }



    /** Get distance between two points in the airspace */
    public static double getDistance(GPoint2D<Double> pt1, GPoint2D<Dou...
...ble> pt2){
        final double earthRadiusNm = 3440.0;    // Radius of the earth ...
...in nautical miles

        /* Equation to calculate distance between two points:
         *cos_AOB = cosd(lat1)*cosd(lat2)*cosd(lon2-lon1)+sind(lat1)*si...
...nd(lat2)
         *AOB = acos(cos_AOB)
         *dist = AOB*3440
         */
        double cosAngle = Math.cos(Math.toRadians(pt1.getLatitude()))*M...
...ath.cos(Math.toRadians(pt2.getLatitude()))
        *Math.cos(Math.toRadians(pt2.getLongitude()-pt1.getLongitude())...
...)
        +Math.sin(Math.toRadians(pt1.getLatitude()))*Math.sin(Math.toRa...
...dians(pt2.getLatitude()));

        double angle = Math.acos(cosAngle);
        double distance = angle*earthRadiusNm;

        return distance;
    }
```

```
    /** Add statistic for flight level */
    public Airspace addFlightLevelStatistic(int plannedFlightLevel, dou...
...ble flownFlightLevel){
        if(flightLevelStats_.containsKey(plannedFlightLevel)){
            ArrayList<Double> flownFlightLevels = flightLevelStats_.get...
...(plannedFlightLevel);
            flownFlightLevels.add(flownFlightLevel);
        }else{
            ArrayList<Double> flownFlightLevels = new ArrayList<Double>...
...();
            flownFlightLevels.add(flownFlightLevel);
            flightLevelStats_.put(plannedFlightLevel, flownFlightLevels...
...);
        }

        return this;
    }

    /** Get statistics for flight level as a string.
        Format: Planned Altitude, Flown Altitude, Flown - Planned
     */
    public String getFlightLevelStatistics(int plannedFlightLevel){
        //sStats.append(plannedFlightLevel); sStats.append(",");

        if(flightLevelStats_.containsKey(plannedFlightLevel)){
            StringBuilder sStats = new StringBuilder();
            ArrayList<Double> flownFlightLevels = flightLevelStats_.get...
...(plannedFlightLevel);

            for(int i=0; i<flownFlightLevels.size(); i++){
                double ffl_i = flownFlightLevels.get(i);
                sStats.append(plannedFlightLevel); sStats.append(APData...
....COMMA);
                sStats.append(ffl_i); sStats.append(APData.COMMA);
                sStats.append(ffl_i - (double)plannedFlightLevel); sSta...
...ts.append(APData.SLASH_N);
            }
            sStats = null;
            return sStats.toString();
        }else{
            return APData.NULLSTRING;
        }
    }

    /** Adds a computer identifier to a callsign (flight plan) */
    public Airspace addComputerIdentifierToCallsign(String callSign, St...
...ring cid){
```

```
        if(flightComputerIdentifiers_.containsKey(callSign)){
            ArrayList<String> cids = flightComputerIdentifiers_.get(cal...
...lSign);
            if(!cids.contains(cid)){
                cids.add(cid);
            }
        }else{
            ArrayList<String> cids = new ArrayList<String>();
            cids.add(cid);
            flightComputerIdentifiers_.put(callSign, cids);
        }

        return this;
    }

    /** Get the number of computer identifiers (flight plan) for a call...
... sign */
    public int numberComputerIdentifiersForCallsign(String callSign){
        if(flightComputerIdentifiers_.containsKey(callSign)){
            ArrayList<String> cids = flightComputerIdentifiers_.get(cal...
...lSign);
            return cids.size();
        }else{ return 0; }

    }
    public String firstCid(String callSign){
        ArrayList<String> cids = flightComputerIdentifiers_.get(callSig...
...n);
        return cids.get(0);
    }


    /** Add a preferred route. This corrects for the case of invalid ro...
...utes between
        between airport pairs.
     */
    public synchronized Airspace addPreferredRoute(String originAirport...
...Id,
            String destinationAirportId, String route){

        HashMap<String, String> routes = preferredRoutes_.containsKey(o...
...riginAirportId) ?
            preferredRoutes_.get(originAirportId) : new HashMap<String,...
... String>(
                Settings.hashMapDefaultInitialCapacity, Settings.hashMa...
...pLoadFactor);
```

```
        if(!routes.containsKey(destinationAirportId)){
            routes.put(destinationAirportId, route);
        }

        preferredRoutes_.put(originAirportId, routes);

        return this;
    }

    /** Get a preferred route that was added with the method addPreferr...
...edRoute */
    public String getPreferredRoute(String originAirportId, String dest...
...inationAirportId){
        HashMap<String, String> routes = preferredRoutes_.get(originAir...
...portId);

        return routes.get(destinationAirportId);
    }

    /** Check to see if a preferred route exists for the specified airp...
...ort pair */
    public boolean hasPreferredRoute(String originAirportId, String des...
...tinationAirportId){
        if(preferredRoutes_.containsKey(originAirportId)){
            HashMap<String, String> routes = preferredRoutes_.get(origi...
...nAirportId);
            return routes.containsKey(destinationAirportId);
        }else{
            return false;
        }
    }

    /** Trim ArrayLists of airspace elements to free additional space *...
.../
    public Airspace trimArrayLists(){
        Iterator<ACenter> e = this.getCenterValueIterator();
        while(e.hasNext()){
            ACenter center = e.next();
            Iterator<ASector> e2 = center.getSectorValueIterator();
            while(e2.hasNext()){
                ASector sector = e2.next();
                sector.trimArrayLists();
            }
        }

        Iterator<String> e3 = this.getVictorAirwayIterator();
        while(e3.hasNext()){
```

```
            Airway awy = this.getVictorAirway(e3.next());
            awy.trimArrayLists();
        }

        Iterator<String> e4 = this.getJetRouteIterator();
        while(e4.hasNext()){
            Airway awy = this.getJetRoute(e4.next());
            awy.trimArrayLists();
        }

        Iterator<String> e5 = this.getRnavRouteIterator();
        while(e5.hasNext()){
            Airway awy = this.getRnavRoute(e5.next());
            awy.trimArrayLists();
        }

        Iterator<Flight> e6 = this.flights_.values().iterator();
        while(e6.hasNext()){
            Flight f = e6.next();
            FTrajectory t = f.getPlannedTrajectory();
            t.trimArrayLists();
        }

        return this;
    }

    public Airspace setNoSectors(int noSectors){
        noSectors_ = noSectors;
        return this;
    }

    public int getNoSectors(){
        return noSectors_;
    }


    public Airspace addFlightToFlightFilter(String flightId){
        if(!flightFilter_.contains(flightId)) flightFilter_.add(flightI...
...d);

        return this;
    }

    public boolean flightInFlightFilter(String flightId){
        return flightFilter_.contains(flightId);
    }
```

```
    public Airspace addFlightDepartureTime(String flightId, String depa...
...rtureTimeString){
        flightDepartureTimes_.put(flightId, departureTimeString);
        return this;
    }

    public boolean hasFlightDepartureTime(String flightId){
        return flightDepartureTimes_.containsKey(flightId);
    }

    public String getFlightDepartureTime(String flightId){
        return flightDepartureTimes_.get(flightId);
    }
}
```

Airway.java

```
/*
 * Airway.java
 *
 * Created on August 7, 2006, 10:56 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
/**
 *
 * @author Jeff
 */
import java.util.*;
public class Airway {

    private static int maxLocationIndex_ = 0;

    private String airwayDesignator_;
    private ArrayList<GPoint2D<Double>> points_;
    private ArrayList<String> pointIdentifiers_;
    private HashMap<String, Integer> order_;
    private int orderInt_;
    private HashMap<String, HashMap<String, AirwayDemand>> airwayDemand...
...._;
    private HashMap<String, Integer> locationIndex_;

    /** Creates a new instance of Airway */
    public Airway(String airwayDesignator) {
        airwayDesignator_ = airwayDesignator;
        points_ = new ArrayList<GPoint2D<Double>>();
        pointIdentifiers_ = new ArrayList<String>();
        order_ = new HashMap<String, Integer>(
                Settings.hashMapDefaultInitialCapacity,
                Settings.hashMapLoadFactor);
        orderInt_ = 0;
        airwayDemand_ = new HashMap<String, HashMap<String, AirwayDeman...
...d>>(
                Settings.hashMapDefaultInitialCapacity,
                Settings.hashMapLoadFactor);
        locationIndex_ = new HashMap<String, Integer>(
                Settings.hashMapDefaultInitialCapacity,
                Settings.hashMapLoadFactor);
    }

    /* Get the airway designator */
    public String getAirwayDesignator(){
```

```
        return airwayDesignator_;
    }

    /** Get the index of the identifier on airway */
    public int getPointIndex(String identifier){
        if(order_.containsKey(identifier)){
            return order_.get(identifier);
        }else{
            for(String loc:order_.keySet()){
                // Try international identifiers
                if(loc.contains("_")){
                    String loc2 = loc.split("_")[0];
                    if(order_.containsKey(loc2)){
                        return order_.get(loc);
                    }
                }
            }
        }
        // Location was not found
        return -1;
    }

    /* Add a point without a designator */
    public Airway addPoint(GPoint2D<Double> pt){
        points_.add(pt);
        pointIdentifiers_.add(APData.NULLSTRING);
        orderInt_++;
        return this;
    }

    /* Add a point with a designator */
    public Airway addPoint(GPoint2D<Double> pt, String ptId){
        points_.add(pt);
        pointIdentifiers_.add(ptId);
        order_.put(ptId, orderInt_++);
        return this;
    }

    public Airway addPointAfter(GPoint2D<Double> pt, String ptIdToAdd, ...
...String previousPtId){
        this.addPointBeforeAfter(pt, ptIdToAdd, previousPtId, true);
        return this;
    }

    private Airway addPointBeforeAfter(GPoint2D<Double> pt, String ptId...
...ToAdd,
            String previousPtId, boolean addPointAfter){
```

```
        int addAtIndex = 0;

        if(addPointAfter){
            // Get index of previous point
            int indexOfPreviousPoint = this.getPointIndex(previousPtId)...
...;
            if(indexOfPreviousPoint == -1) return this;
            addAtIndex = indexOfPreviousPoint + 1;
        }else{  // addPointBefore
            // Get index of next point
            int indexOfNextPoint = this.getPointIndex(previousPtId);
            if(indexOfNextPoint == -1) return this;
            addAtIndex = indexOfNextPoint;
        }

        // Add the point
        points_.add(addAtIndex, pt);
        pointIdentifiers_.add(addAtIndex, ptIdToAdd);
        order_.put(ptIdToAdd, addAtIndex);

        // Update the indices of the other points
        for(int i=addAtIndex+1; i<points_.size(); i++){
            String id = pointIdentifiers_.get(i);
            order_.put(id, i);
        }

        return this;
    }

    public Airway addPointBefore(GPoint2D<Double> pt, String ptIdToAdd,...
... String nextPtId){
        this.addPointBeforeAfter(pt, ptIdToAdd, nextPtId, false);
        return this;
    }

    /* Get point based on point designator */
    public GPoint2D<Double> getPoint(String ptId){
        int idx = order_.get(ptId);
        return points_.get(idx);
    }

    /* Get the points between two NAVAID locations.
       Note: excludes the starting and ending point in the arraylist.
     */
    public ArrayList<String> getPointIdentifiers(String startPoint, Str...
...ing endPoint){
        ArrayList<String> awyPointIdentifiers = new ArrayList<String>()...
```

```
...;

        int startPtIdx = order_.get(startPoint);
        int endPtIdx = order_.get(endPoint);

        // Return an empty array if start point and end point are adjac...
...ent
        if(Math.abs(startPtIdx - endPtIdx) <= 1){
            return awyPointIdentifiers;
        }

        if(startPtIdx < endPtIdx){
            for(int i = startPtIdx+1; i < endPtIdx; i++){
                awyPointIdentifiers.add(pointIdentifiers_.get(i));
            }
        }else{  // endPtIdx < startPtIdx
            for(int i = startPtIdx -1; i > endPtIdx; i--){
                awyPointIdentifiers.add(pointIdentifiers_.get(i));
            }
        }


        return awyPointIdentifiers;
    }

    public String searchForAirwayIntersection(Airway intersectingAirway...
...){
        for(int i=0; i<intersectingAirway.noPoints(); i++){
            String candidateLocation = intersectingAirway.getPointIdent...
...ifier(i);
            if(order_.containsKey(candidateLocation)){
                return candidateLocation;
            }
        }
        return APData.NULLSTRING;
    }

    public boolean onAirway(ArrayList<String> identifiers){
        // Special case of only two identifiers
        if(identifiers.size() == 2)
            return order_.containsKey(identifiers.get(0)) &&
                    order_.containsKey(identifiers.get(1));

        // Check if the identifiers are on the airway
        boolean lastIdentifierOnAirway = false;
        for(String identifiers_i:identifiers){
            if(identifiers_i.length() > 0){
```

```
                boolean currentIdentifierOnAirway = order_.containsKey(...
...identifiers_i);

                // Check if two consecutive identifiers are on the airw...
...ay
                if(lastIdentifierOnAirway && currentIdentifierOnAirway)...
...                     return true;

                lastIdentifierOnAirway = currentIdentifierOnAirway;
            }
        }

        return false;
    }

    /* Check if a specific NAVAID, fix, or international NAVAID/FIX is ...
...on airway */
    public String onAirway(String identifier){
        // Check point identifier as is
        if(pointIdentifiers_.contains(identifier)){
            return APData.TRUE;
        }

        // Check for international point identifiers (e.g. check XXX_CN...
... as XXX)
        for(String id:pointIdentifiers_){
            int indexOfUnderscore = id.indexOf(APData.UNDERSCORE);
            String idToUnder = indexOfUnderscore > -1 ?
                id.substring(0, indexOfUnderscore) :
                id;
            //String[] idSplit = id.split(APData.UNDERSCORE);
            if(identifier.equals(idToUnder)){
                //idSplit = null;
                idToUnder = null;
                return id;
            }
            idToUnder = null;
            //idSplit = null;
        }

        return APData.FALSE;
    }

    public boolean identifierOnAirway(String identifier){
        return pointIdentifiers_.contains(identifier);
    }
```

```
    /* Get point based on location in array */
    public GPoint2D<Double> getPoint(int idx){
        return points_.get(idx);
    }

    public String getPointIdentifier(int idx){
        return pointIdentifiers_.get(idx);
    }

    /* Number of points that define the airway */
    public int noPoints(){
        return points_.size();
    }

    public Airway addAirwayDemand(String fromNode, String toNode,
            GPoint2D<Double> fromNodeLocation, GPoint2D<Double> toNodeL...
...ocation,
            int demand){

        HashMap<String, AirwayDemand> toNodeDemand;

        if(airwayDemand_.containsKey(fromNode)){
            toNodeDemand = airwayDemand_.get(fromNode);
        }else{
            toNodeDemand = new HashMap<String, AirwayDemand>(
                    Settings.hashMapDefaultInitialCapacity,
                    Settings.hashMapLoadFactor);
            airwayDemand_.put(fromNode, toNodeDemand);
        }

        if(toNodeDemand.containsKey(toNode)){
            AirwayDemand awyDemand = toNodeDemand.get(toNode);
            awyDemand.addDemand(1);
        }else{
            AirwayDemand awyDemand = new AirwayDemand(fromNode, toNode,...
...
                fromNodeLocation, toNodeLocation, demand);
            toNodeDemand.put(toNode, awyDemand);
        }

        if(!locationIndex_.containsKey(fromNode)){
            locationIndex_.put(fromNode, new Integer(maxLocationIndex_+...
...+));
        }

        if(!locationIndex_.containsKey(toNode)){
```

```
            locationIndex_.put(toNode, new Integer(maxLocationIndex_++)...
...);
        }

        return this;
    }

    public ArrayList<AirwayDemand> getDemandList(){
        ArrayList<AirwayDemand> allAirwayDemand = new ArrayList<AirwayD...
...emand>();

        Iterator<String> e = airwayDemand_.keySet().iterator();
        while(e.hasNext()){
            String from = e.next();
            allAirwayDemand.addAll(airwayDemand_.get(from).values());
        }

        return allAirwayDemand;
    }

    public int getLocationIndex(String location){
        return locationIndex_.get(location);
    }

    public boolean adjacentIdentifiers(String location1, String locatio...
...n2){
        if(!locationIndex_.containsKey(location1) ||
                !locationIndex_.containsKey(location2)) return false;

        int firstIdx = locationIndex_.get(location1);
        int secondIdx = locationIndex_.get(location2);

        // Index will differ by 1 if the identifiers are adjacent
        if(Math.abs(firstIdx - secondIdx) == 1){
            return true;
        }else{
            return false;
        }
    }

    public Airway trimArrayLists(){
        points_.trimToSize();
        pointIdentifiers_.trimToSize();
        return this;
    }

}
```

AirwayDemand.java

```
/*
 * AirwayDemand.java
 *
 * Created on April 4, 2007, 3:49 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author jeff
 */
import java.util.*;
public class AirwayDemand {
    private String fromNode_;
    private String toNode_;
    private GPoint2D<Double> fromNodeLocation_;
    private GPoint2D<Double> toNodeLocation_;
    private int demand_;

    /** Creates a new instance of AirwayDemand */
    public AirwayDemand(String fromNode, String toNode,
            GPoint2D<Double> fromNodeLocation, GPoint2D<Double> toNodeL...
...ocation,
            int demand) {
        fromNode_ = fromNode;
        toNode_ = toNode;
        fromNodeLocation_ = fromNodeLocation;
        toNodeLocation_ = toNodeLocation;
        demand_ = demand;
    }

    public String getFromNode(){
        return fromNode_;
    }

    public String getToNode(){
        return toNode_;
    }

    public GPoint2D<Double> getFromNodeLocation(){
        return fromNodeLocation_;
    }

    public GPoint2D<Double> getToNodeLocation(){
```

```
        return toNodeLocation_;
    }

    public int getDemand(){
        return demand_;
    }

    public AirwayDemand addDemand(int additionalDemand){
        demand_ += additionalDemand;

        return this;
    }
}
```

APData.java

```
//
//  APData.java
//  AirPlanJ
//
//  Created by jeff on 8/2/06.
//  Copyright 2006. All rights reserved.
//
import java.util.*;
import java.util.prefs.*;
public class APData {
    static public Airspace airspace;
    static public Wind wind;
    static public Properties properties;
    static public Network network;
    static public HashMap<Integer, ConvectionForecast> convectionForeca...
...sts;
    static public HashMap<String, KernelDensity> kernelDensities;
    static public ConvectionForecastDistribution convectionForecastDist...
...ribution;
    static public MatlabDriver matlabDriver;
    /** String constants */
    public static final String COMMA = ",";
    public static final String UNDERSCORE = "_";
    public static final String SPACE = " ";
    public static final String COLON = ":";
    public static final String PERIOD = ".";
    public static final String BAR = "|";
    public static final String FORWARDSLASH = "/";
    public static final String PLUS = "+";
    public static final String SLASH_N = "\n";
    public static final String ARPT = "ARPT";
    public static final String COORDN_ = "COORDN-";
    public static final String GPS_WP = "GPS-WP";
    public static final String MIL_REP = "MIL-REP";
    public static final String MIL_WAY = "MIL-WAY";
    public static final String NRS_WAY = "NRS-WAY";
    public static final String RADAR = "RADAR";
    public static final String RNAV_WP = "RNAV-WP";
    public static final String WAYPOIN = "WAYPOIN";
    public static final String US = "US";
    public static final String NAVAID = "NAVAID";
    public static final String FIX = "FIX";
    public static final String ZERO = "0";
    public static final String FIVE = "5";
    public static final String NULLSTRING = "";
    public static final String A = "A";
```

```
    public static final String B = "B";
    public static final String C = "C";
    public static final String D = "d";
    public static final String E = "E";
    public static final String G = "G";
    public static final String J = "J";
    public static final String K = "K";
    public static final String M = "m";
    public static final String N = "N";
    public static final String Q = "Q";
    public static final String R = "R";
    public static final String T = "T";
    public static final String V = "V";
    public static final String W = "W";
    public static final String X = "X";
    public static final String CLIMB = "Climb";
    public static final String DESCENT = "Descent";
    public static final String UNKNOWN = "UNKNOWN";
    public static final String PTF = "PTF";
    public static final String AC_TYPE = "AC/Type";
    public static final String FGTH = "FGTH";
    public static final String FGTL = "FGTL";
    public static final String FGTN = "FGTN";
    public static final String NIMBUSTR = "NimbusTR";
    public static final String SR22G2 = "SR22G2";
    public static final String VLJ = "VLJ";
    public static final String FBUS31 = "FBUS31";
    public static final String FBUS = "FBUS";
    public static final String VALID = "VALID";
    public static final String _FLOOR_ = "_FL_";
    public static final String _CENTROID_ = "_CD_";
    public static final String _CEILING_ = "_CG_";
    public static final String _BOUNDARY_ = "_BD_";
    public static final String _APT = "_APT";
    public static final String NULL = "NULL";
    public static final String VFR = "VFR";
    public static final String OTP = "OTP";
    public static final String ABV = "ABV";
    public static final String UTC = "UTC";
//        public static final String INTRASECTOR = "Intrasector";
    public static final String TRUE = "true";
    public static final String FALSE = "false";
//        public static final String ADDITIONAL_LINKS = "Additional Lin...
...ks";
//        public static final String VICTOR_AIRWAY = "Victor Airway";
//        public static final String AIRPORT_CONNECTOR = "Airport Conne...
...ctor";
```

```
//        public static final String JET_ROUTE = "Jet Route";
//        public static final String RNAV_ROUTE = "RNAV Route";
//        public static final String FIX_L = "Fix";
//        public static final String NAVAID_L = "Navaid";
    public static final String ARCLENGTH = "arclength";

    public static void Initialize() {
        airspace = new Airspace();
        wind = new Wind();
        properties = new Properties();
        network = new Network();
        convectionForecasts = new HashMap<Integer, ConvectionForecast>(...
...);
        kernelDensities = new HashMap<String, KernelDensity>();
        convectionForecastDistribution = new ConvectionForecastDistribu...
...tion();
    }
    public static void InitializeMatlab() {
        // Initialize MATLAB
        System.loadLibrary("jnimatlabdriver");
        System.loadLibrary("libmatlab");

        matlabDriver = new MatlabDriver();
        matlabDriver.applicationInitialize();
        matlabDriver.libInitialize();
    }
    public static void TerminateMatlab() {
        // Terminate the MATLAB library
        matlabDriver.applicationTerminate();
        matlabDriver.libTerminate();

        matlabDriver = null;
    }
}
```

ASector.java

```
/*
 * ASector.java
 *
 * Created on July 24, 2006, 9:46 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
/**
 *
 * @author jeff
 */
import java.util.*;
public class ASector {
    /** Initial capacity for number of modules that defines a sector */...
...    static final int INIT_NUM_MODULES = 5;

    /** Initial capacity for number of locations that an airway interse...
...cts
     *  with a sector boundary.
     */
    static final int INIT_NUM_AIRWAY_INTERSECTIONS = 20;

    /** RAMS Default Data */
    static final int DISTANCE_EXTENDED_ALTITUDE = 20;
    static final int DISTANCE_EXTENDED_BORDER = 10;
    static final int TACTICAL_RADAR_ENTRY_WHEN_NO_SECTOR_PIERCE = 0;

    static final int LATERAL_SEPARATION_DISTANCE = 5;   // nm
    static final int LATERAL_SEPARATION_TIME = 0;       // seconds
    static final int LONGITUDE_SEPARATION_DISTANCE = 5; // nm
    static final int LONGITUDE_SEPARATION_TIME = 0;     // seconds

    private String name_;
    private ArrayList <ASectorModule> modules_;

    /* Data structure to hold demand locations. */
    private ArrayList <GPoint2D<Double>> demandLocations_;
    private ArrayList <ArrayList<Integer>> demand_;

    /* Representation of a null sector */
    public static final ASector NullSector = new ASector(APData.NULL);


    /* List of NAVAIDs and Fixes in sector */
    private HashMap<String,Double> navaidsInSectorFloor_;
```

```
    private HashMap<String,Double> navaidsInSectorCeiling_;
    private HashMap<String,Double> fixesInSectorFloor_;
    private HashMap<String,Double> fixesInSectorCeiling_;

    /* List of NAVAIDs and Fixes on airways that cross the sector */
    private HashMap<String,Double> waypointOnAwyOutsideSector_;

    /* Location of airway intersections with sector boundary. Key is th...
...e
       identifier of the previous node on the jet route contained in
       navaidsInSector_ or fixesInSector_ .*/
    private HashMap<String,HashMap<String,GPoint3D<Double>>> airwayInte...
...rsections_;


    /* Monitor Alert Parameter (MAP) value for the sector.  */
    private int mapValue_;


    /** Create an index for arrays to store capacity and flow */
    private int arrayIndex_;

    /** Creates a new instance of ASector */
    //public ASector() {
    //}

    private double minLatitude_;
    private double maxLatitude_;
    private double minLongitude_;
    private double maxLongitude_;

    public ASector(String name){
        name_ = name;
        modules_ = new ArrayList <ASectorModule>(INIT_NUM_MODULES);

        demandLocations_ = new ArrayList<GPoint2D<Double>>(1);
        demand_ = new ArrayList<ArrayList<Integer>>(1);

        navaidsInSectorFloor_ = new HashMap<String,Double>(
                Settings.hashMapDefaultInitialCapacity,
                Settings.hashMapLoadFactor);
        navaidsInSectorCeiling_ = new HashMap<String,Double>(
                Settings.hashMapDefaultInitialCapacity,
                Settings.hashMapLoadFactor);
        fixesInSectorFloor_ = new HashMap<String,Double>(
                Settings.hashMapDefaultInitialCapacity,
                Settings.hashMapLoadFactor);
```

```
        fixesInSectorCeiling_ = new HashMap<String,Double>(
                Settings.hashMapDefaultInitialCapacity,
                Settings.hashMapLoadFactor);

        waypointOnAwyOutsideSector_ = new HashMap<String,Double>(
                Settings.hashMapDefaultInitialCapacity,
                Settings.hashMapLoadFactor);

        airwayIntersections_ = new HashMap<String, HashMap<String,GPoin...
...t3D<Double>>>(
                Settings.hashMapDefaultInitialCapacity,
                Settings.hashMapLoadFactor);

        mapValue_ = 0;

        arrayIndex_ = -1;

        minLatitude_ = Double.MAX_VALUE;
        maxLatitude_ = Double.MIN_VALUE;
        minLongitude_ = Double.MAX_VALUE;
        maxLongitude_ = Double.MIN_VALUE;
    }

    private ASector calculateMinMaxLatLon(){
        for(ASectorModule mod:modules_){
            double[] lats = mod.getNodeLatitudes();
            double[] lons = mod.getNodeLongitudes();

            for(int i=0; i<lats.length; i++){
                if(lats[i] < minLatitude_) minLatitude_ = lats[i];
                if(lats[i] > maxLatitude_) maxLatitude_ = lats[i];

                if(lons[i] < minLongitude_) minLongitude_ = lons[i];
                if(lons[i] > maxLongitude_) maxLongitude_ = lons[i];
            }
        }
        return this;
    }

    public double getMinimumLat(){
        if(minLatitude_ == Double.MAX_VALUE) calculateMinMaxLatLon();

        return minLatitude_;
    }

    public double getMaximumLat(){
        if(maxLatitude_ == Double.MIN_VALUE) calculateMinMaxLatLon();
```

```
        return maxLatitude_;
    }

    public double getMinimumLon(){
        if(minLongitude_ == Double.MAX_VALUE) calculateMinMaxLatLon();

        return minLongitude_;
    }

    public double getMaximumLon(){
        if(maxLongitude_ == Double.MIN_VALUE) calculateMinMaxLatLon();

        return maxLongitude_;
    }

    public ASector setArrayIndex(int arrayIndex){
        arrayIndex_ = arrayIndex;
        return this;
    }

    public int getArrayIndex(){
        return arrayIndex_;
    }

    /** Add a point of intersection with a jet route/airway */
    public synchronized ASector addAirwayIntersection(String wayPoint1,...
... String wayPoint2,
            double longitude, double latitude, double elevation){
        String fromWaypoint = null;
        String toWaypoint = null;
        if(navaidsInSectorFloor_.containsKey(wayPoint1) ||
                fixesInSectorFloor_.containsKey(wayPoint1)){
//            if(navaidsInSectorFloor_.containsKey(wayPoint2) ||
//                    fixesInSectorFloor_.containsKey(wayPoint2)){
//                // If both wayPoint1 and wayPoint2 in sector do not a...
...dd
//                return this;
//            }else{
              fromWaypoint = wayPoint1;
              toWaypoint = wayPoint2;
//            }
        }else if(navaidsInSectorFloor_.containsKey(wayPoint2) ||
                    fixesInSectorFloor_.containsKey(wayPoint2)){
            //
            fromWaypoint = wayPoint2;
            toWaypoint = wayPoint1;
```

```
        }else{
            /* Both waypoints outside the sector.
               Use first waypoint as the from waypoint and add to list ...
...of
             * navaids and fixes outside the sector.
             */
            fromWaypoint = wayPoint1;
            toWaypoint = wayPoint2;


            if(!waypointOnAwyOutsideSector_.containsKey(wayPoint1)){
                waypointOnAwyOutsideSector_.put(wayPoint1,elevation);
            }

            if(!waypointOnAwyOutsideSector_.containsKey(wayPoint2)){
                waypointOnAwyOutsideSector_.put(wayPoint2,elevation);
            }
        }

        boolean hasFromWaypoint = airwayIntersections_.containsKey(from...
...Waypoint);

        HashMap<String, GPoint3D<Double>> locs = hasFromWaypoint ?
            airwayIntersections_.get(fromWaypoint) :
            new HashMap<String, GPoint3D<Double>>(Settings.hashMapDefau...
...ltInitialCapacity,
                Settings.hashMapLoadFactor);

        boolean hasToWaypoint = locs.containsKey(toWaypoint);

        if(!hasToWaypoint){
            elevation = 0.5D*(this.getFloor() + this.getCeiling());
            GPoint3D<Double> pt = new GPoint3D<Double>(longitude, latit...
...ude, elevation);
            locs.put(toWaypoint, pt);
            if(!hasFromWaypoint){
                airwayIntersections_.put(fromWaypoint, locs);
            }
        }
        return this;
    }

    public Iterator<String> getAirwayIntersectionIterator(){
        return airwayIntersections_.keySet().iterator();
    }

    /** Iterator of waypoints outside of the sector (i.e. boundary cros...
```

```
...sings) */
    public Iterator<String> getToWaypointIntersectionIterator(String fr...
...omWaypoint){
        HashMap<String, GPoint3D<Double>> toWaypoints =
                airwayIntersections_.get(fromWaypoint);
        return toWaypoints.keySet().iterator();
    }

    /** Get location where airway intersects with the sector boundary. ...
...        Note that fromWaypoint is within the sector boundary and toWayp...
...oint
        is outside the sector boundary.
     */
    public GPoint3D<Double> getAirwayIntersectionLocation(String fromWa...
...ypoint,
            String toWaypoint){
        HashMap<String, GPoint3D<Double>> locations =
                airwayIntersections_.get(fromWaypoint);
        return locations.get(toWaypoint);
    }


    /** The maximum elevation of all the sector modules */
    public int getCeiling(){
        // Unlimited ceiling for the null (external) sector
        if(name_.equals(APData.NULL)) return 999;

        int maxCeiling = 0;
        for (int i = 0; i < modules_.size(); i++){
            if(modules_.get(i).getCeiling() > maxCeiling){
                maxCeiling = modules_.get(i).getCeiling();
            }
        }
        return maxCeiling;
    }

    /** The minimum elevation of all the sector modules */
    public int getFloor(){
        // A floor of zero for the null (external) sector
        if(name_.equals(APData.NULL)) return 0;

        int minFloor = 999;
        for (int i = 0; i < modules_.size(); i++){
            if(modules_.get(i).getFloor() < minFloor){
                minFloor = modules_.get(i).getFloor();
            }
        }
```

```
        return minFloor;
    }

    /** Add a sector module */
    public ASector addModule(ASectorModule module){
        modules_.add(module);
        return this;
    }

    /** Number of modules used to iterate through module list
     * using getModule method */
    public int noModules(){
        return modules_.size();
    }

    /** Get a sector module by index */
    public ASectorModule getModule(int index){
        return modules_.get(index);
    }

    public ASectorModule[] getModules(){
        ASectorModule[] moduleArray = new ASectorModule[modules_.size()...
...];
        for(int i=0; i< modules_.size(); i++){
            moduleArray[i] = modules_.get(i);
        }
        return moduleArray;
    }

    /** Get sector name */
    public String getName(){
        return name_;
    }


    /* Add demand at a point in the sector (entering or exiting) by hou...
...r */
    public ASector addDemandAtPoint(GPoint2D<Double> pt, int hour){
        // Check if any location is within 3nm
        for(int i = 0; i < demandLocations_.size(); i++){
            GPoint2D<Double> pt2 = demandLocations_.get(i);

            double distance = Airspace.getDistance(pt, pt2);

            // Point was found, return.
            if(distance < 3.0){
                ArrayList<Integer> demand = demand_.get(i);
```

```
            int prevDemand = demand.get(hour);
            demand.set(hour, prevDemand+1);
            return this;
        }
    }

    // Point not found, add another
    demandLocations_.add(pt);
    ArrayList<Integer> demand = new ArrayList<Integer>(25);
    for(int i=0; i<25; i++){
        demand.add(0);
    }
    demand.set(hour,1);
    demand_.add(demand);

    return this;
}

/* Get hourly demand at the demand point */
public int getHourlyDemandAtPoint(GPoint2D<Double> pt, int hour){
    for(int i = 0; i < demandLocations_.size(); i++){
        GPoint2D<Double> pt2 = demandLocations_.get(i);

        double distance = Airspace.getDistance(pt, pt2);

        if(distance < 3.0){
            ArrayList<Integer> demand = demand_.get(i);
            return demand.get(hour);
        }
    }

    return 0;
}

public double getHourlyDemand(int hour){
    double totalDemand = 0;
    for(int i = 0; i < demandLocations_.size(); i++){
        totalDemand += demand_.get(i).get(hour);
    }

    return totalDemand/2.0D;
}

public Iterator<String> getNavaidsInSectorIterator(){
    return navaidsInSectorFloor_.keySet().iterator();
}
```

```
    public synchronized ASector addNavaidToSector(String identifier, do...
...uble moduleFloor,
            double moduleCeiling){
        if(navaidsInSectorFloor_.containsKey(identifier)){
            double currentMinModuleFloor = navaidsInSectorFloor_.get(id...
...entifier);
            if(moduleFloor < currentMinModuleFloor) navaidsInSectorFloo...
...r_.put(identifier, moduleFloor);
        }else{
            navaidsInSectorFloor_.put(identifier, moduleFloor);
        }

        if(navaidsInSectorCeiling_.containsKey(identifier)){
            double currentMaxModuleCeiling = navaidsInSectorCeiling_.ge...
...t(identifier);
            if(moduleCeiling > currentMaxModuleCeiling) navaidsInSector...
...Ceiling_.put(identifier, moduleCeiling);
        }else{
            navaidsInSectorCeiling_.put(identifier, moduleCeiling);
        }
        return this;
    }

    public boolean hasNavaid(String identifier){
        return navaidsInSectorFloor_.containsKey(identifier);
    }

    public double getNavaidModuleFloor(String identifier){
        return navaidsInSectorFloor_.get(identifier);
    }

    public double getNavaidModuleCeiling(String identifier){
        return navaidsInSectorCeiling_.get(identifier);
    }

    public Iterator<String> getFixesInSectorIterator(){
        return fixesInSectorFloor_.keySet().iterator();
    }

    public synchronized ASector addFixToSector(String identifier, doubl...
...e moduleFloor,
            double moduleCeiling){
        if(fixesInSectorFloor_.containsKey(identifier)){
            double currentMinModuleFloor = fixesInSectorFloor_.get(iden...
...tifier);
            if(moduleFloor < currentMinModuleFloor) fixesInSectorFloor_...
....put(identifier, moduleFloor);
```

```
        }else{
            fixesInSectorFloor_.put(identifier, moduleFloor);
        }

        if(fixesInSectorCeiling_.containsKey(identifier)){
            double currentMaxModuleCeiling = fixesInSectorCeiling_.get(...
...identifier);
            if(moduleCeiling > currentMaxModuleCeiling) fixesInSectorCe...
...iling_.put(identifier, moduleCeiling);
        }else{
            fixesInSectorCeiling_.put(identifier, moduleCeiling);
        }

        return this;
    }

    public boolean hasFix(String identifier){
        return fixesInSectorFloor_.containsKey(identifier);
    }

    public double getFixModuleFloor(String identifier){
        return fixesInSectorFloor_.get(identifier);
    }

    public double getFixModuleCeiling(String identifier){
        return fixesInSectorCeiling_.get(identifier);
    }

    /* If a Navaid and a Fix with the same ID are listed in the sector ...
...only the
     * fix is retained. Returns number of duplicates removed.
     */
    public int removeDuplicateNavaidsFixes(){
        int numDuplicates = 0;

        Iterator<String> e = fixesInSectorFloor_.keySet().iterator();
        while(e.hasNext()){
            String fixId = e.next();
            if(navaidsInSectorFloor_.containsKey(fixId)){
                navaidsInSectorFloor_.remove(fixId);
                navaidsInSectorCeiling_.remove(fixId);
                numDuplicates++;
            }
        }

        return numDuplicates;
    }
```

```
    public int getMapValue(){
        return mapValue_;
    }

    public ASector setMapValue(int mapValue){
        mapValue_ = mapValue;
        return this;
    }

    // Reduce memory requirement by trimming arraylists
    public ASector trimArrayLists(){
        modules_.trimToSize();

        for(ASectorModule module:modules_){
            module.trimArrayLists();
        }

        return this;
    }



}
```

ASectorModule.java

```
/*
 * ASectorModule.java
 *
 * Created on August 11, 2006, 9:19 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
/**
 *
 * @author jeff
 */
import java.util.*;
public class ASectorModule {
    /** Initial capacity for number of nodes that defines a sector modu...
...le */
    static final int INIT_NUM_NODES = 50;

    private String name_;
    private int id_;
    private int floor_;
    private int ceiling_;
    private ArrayList <GPoint2D <Double>> nodes_;

    private double[] nodeLatitudes_;
    private double[] nodeLongitudes_;
    private boolean nodeArraysSet_;

    private double area_;

    /** Creates a new instance of ASectorModule */
    public ASectorModule(String name, int id, int floor, int ceiling){
        name_ = name;
        id_ = id;
        floor_ = floor;
        ceiling_ = ceiling+1;   // Add one to ensure that no gaps exist...
... between modules
        nodes_ = new ArrayList <GPoint2D <Double>>(INIT_NUM_NODES);
        nodeArraysSet_ = false;
        area_ = -1.0D;
    }

    public synchronized ASectorModule setArea(double area){
        area_ = area;

        return this;
```

```
}

public synchronized double getArea(){
    return area_;
}

/** Add a node that defines a sector module */
public ASectorModule addNode(GPoint2D<Double> node){
    nodes_.add(node);
    return this;
}

/** Number of nodes used to iterate through node list
 * using getNode method */
public int noNodes(){
    return nodes_.size();
}

/** Get a node of the sector module by index */
public GPoint2D<Double> getNode(int index){
    return nodes_.get(index);
}

public GPoint2D[] getNodes(){
    GPoint2D[] nodesArray = new GPoint2D[nodes_.size()];
    for(int i=0; i<nodes_.size(); i++){
        nodesArray[i] = nodes_.get(i);
    }
    return nodesArray;
}

public double[] getNodeLatitudes(){
    if(nodeArraysSet_){
        return nodeLatitudes_;
    }else{
        nodeLatitudes_ = new double[nodes_.size()];
        for(int i=0; i<nodes_.size(); i++){
            nodeLatitudes_[i] = nodes_.get(i).getLatitude();
        }

        getNodeLongitudes();

        nodeArraysSet_ = true;
        return nodeLatitudes_;
    }
}
```

```
    public double[] getNodeLongitudes(){
        if(nodeArraysSet_){
            return nodeLongitudes_;
        }else{
            nodeLongitudes_ = new double[nodes_.size()];
            for(int i=0; i<nodes_.size(); i++){
                nodeLongitudes_[i] = nodes_.get(i).getLongitude();
            }
            return nodeLongitudes_;
        }
    }



    /** Get sector module name */
    public String getName(){
        return name_;
    }

    /** Get sector module identification number */
    public int getId(){
        return id_;
    }

    /** Get sector module floor in Flight Level units   */
    public int getFloor(){
        return floor_;
    }

    /** Get sector module ceiling in Flight Level units */
    public int getCeiling(){
        return ceiling_;
    }

    public ASectorModule trimArrayLists(){
        nodes_.trimToSize();
        return this;
    }

}
```

ChangeAltitude.java

```
/*
 * ChangeAltitude.java
 *
 * Created on October 21, 2007, 8:32 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package MATLAB;
/**
 *
 * @author Jeff
 */
import java.util.*;
public class ChangeAltitude {
    private double currentTime_;
    private double currentDistance_;
    private double currentAltitude_;
    private Node3D<Double> toNode_;
    private Scenario scenario_;
    private Flight flight_;

    /** Creates a new instance of ChangeAltitude */
    public ChangeAltitude(double currentTime, double currentDistance,
            double currentAltitude, Node3D<Double> toNode,
            Scenario scenario, Flight flight) {
        currentTime_ = currentTime;
        currentDistance_ = currentDistance;
        currentAltitude_ = currentAltitude;
        toNode_ = toNode;
        scenario_ = scenario;
        flight_ = flight;
    }

    public Node3D<Double> getToNode(){
        return toNode_;
    }

    private boolean sameAirspaceIdentifier(Link3D link){
        Node3D<Double> fromNode = link.getFromNode();
        Node3D<Double> toNode = link.getToNode();
        if(fromNode.getAirspaceIdentifier().equals(toNode.getAirspaceId...
...entifier()))
            return true;

        return false;
```

```
    }

    /** Change altitude calculations */
    public ChangeAltitude run(TraversalTimeDistribution ttd){
        // Get list of links exiting to node
        ArrayList<Link3D> outLinks2 = toNode_.getOutLinkList();

        double linkCeilingAltitude = scenario_.getLastLinkCeilingAltitu...
...de(flight_.getCallSign());
        double linkFloorAltitude = scenario_.getLastLinkFloorAltitude(f...
...light_.getCallSign());

        for(Link3D outLink2:outLinks2){
            toNode_ = outLink2.getToNode();

            // Check that the to and from nodes are at the same airspac...
...e identifier
            if(!sameAirspaceIdentifier(outLink2)) continue;

            // Check if the to node of the link goes to the ceiling
            // or to the floor.
            boolean isCeiling = toNode_.isAtCeiling();
            boolean isFloor = toNode_.isAtFloor();

            if(isCeiling && (currentAltitude_ > linkCeilingAltitude) ||...
...                    isFloor && (currentAltitude_ < linkFloorAltitude) |...
...|
                    (currentAltitude_ == 0.0D && (isCeiling || isFloor)...
... &&
                    (outLink2.getToNode().getElevation() < outLink2.get...
...FromNode().getElevation())){

                scenario_.addFlightPosition(flight_.getCallSign(), outL...
...ink2.getToNodeId(),
                        outLink2.getToNode(), currentTime_, currentDist...
...ance_, currentAltitude_,
                        ttd);
                scenario_.addFlightLink(flight_.getCallSign(), outLink2...
....getIdentifier(), outLink2);

                // The next link is from Floor-Ceiling or Ceiling-Floor...
...                ArrayList<Link3D> outLinks3 = toNode_.getOutLinkList();...
...
                //if(outLinks3.size() < 2) continue;

                for(Link3D outLink3A:outLinks3){
                    if(!sameAirspaceIdentifier(outLink3A)) continue;
```

```
                    Node3D<Double> toNodeA = outLink3A.getToNode();
                    if(isCeiling && toNodeA.isAtFloor() || isFloor && t...
...oNodeA.isAtCeiling()){
                        toNode_ = toNodeA;
                        scenario_.addFlightPosition(flight_.getCallSign...
...(), outLink3A.getToNodeId(),
                                outLink3A.getToNode(), currentTime_, cu...
...rrentDistance_, currentAltitude_,
                                ttd);
                        scenario_.addFlightLink(flight_.getCallSign(), ...
...outLink3A.getIdentifier(),
                                outLink3A);
                        break;
                    }
                }

                // The next link is from Floor-Centroid or Ceiling-Cent...
...roid
                ArrayList<Link3D> outLinks4 = toNode_.getOutLinkList();...
...
                //if(outLinks4.size() < 2) continue;

                for(Link3D outLink4A:outLinks4){
                    if(!sameAirspaceIdentifier(outLink4A)) continue;

                    Node3D<Double> toNodeA = outLink4A.getToNode();
                    if(toNodeA.isAtCentroid()){
                        toNode_ = toNodeA;
                        scenario_.addFlightPosition(flight_.getCallSign...
...(), outLink4A.getToNodeId(), outLink4A.getToNode(),
                                currentTime_, currentDistance_, current...
...Altitude_, ttd);
                        scenario_.addFlightLink(flight_.getCallSign(), ...
...outLink4A.getIdentifier(), outLink4A);
                        return this;
                    }
                }
            }
        }
        return this;
    }

}
```

ConvectionForecast.java

```
/*
 * ConvectionForecast.java
 *
 * Created on August 22, 2007, 10:59 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author Jeff
 */
import java.util.*;
public class ConvectionForecast {
    // List of exterior point locations (latitude, longitude)
    private ArrayList<ArrayList<GPoint2D<Double>>> convectionForecastGr...
...oupLocations_;

    // List of convection forecast group centroids
    private ArrayList<GPoint2D<Double>> convectionForecastCentroidLocat...
...ions_;

    // List of convection forecast area
    private ArrayList<Double> convectionForecastArea_;

    // Forecasted tops of the convection (kft)
    // 3: 370+
    // 2: 310-370
    // 1: 250-310
    // 0: < 250
    private ArrayList<Integer> convectionForecastTops_;
    private ArrayList<GPoint2D<Double>> convectionForecastTopLocations_...
...;


    // Observed convection polygon separated by NaN.
    private ArrayList<GPoint2D<Double>> observedConvectionPolygon_;

    // 10 nm buffer around the observed polygon. Also separated by NaN....
...    private ArrayList<GPoint2D<Double>> observedConvection10NmBufferPol...
...ygon_;

    // 20 nm buffer around the observed polygon. Also separated by NaN....
...    private ArrayList<GPoint2D<Double>> observedConvection20NmBufferPol...
...ygon_;
```

```
    // Observed tops of the convection
    // Centroid, Observed tops of convection (0 to 3)
    private ArrayList<Integer> observedConvectionTops_;
    private ArrayList<GPoint2D<Double>> observedConvectionTopLocations_...
...;


    /** Creates a new instance of ConvectionForecast */
    public ConvectionForecast() {
        convectionForecastGroupLocations_ = new ArrayList<ArrayList<GPo...
...int2D<Double>>>();
        convectionForecastCentroidLocations_ = new ArrayList<GPoint2D<D...
...ouble>>();
        convectionForecastArea_ = new ArrayList<Double>();
        convectionForecastTops_ = new ArrayList<Integer>();
        convectionForecastTopLocations_ = new ArrayList<GPoint2D<Double...
...>>();
        observedConvectionPolygon_ = new ArrayList<GPoint2D<Double>>();...
...        observedConvection10NmBufferPolygon_ = new ArrayList<GPoint2D<D...
...ouble>>();
        observedConvection20NmBufferPolygon_ = new ArrayList<GPoint2D<D...
...ouble>>();
        observedConvectionTops_ = new ArrayList<Integer>();
        observedConvectionTopLocations_ = new ArrayList<GPoint2D<Double...
...>>();
    }



    public ConvectionForecast addConvectionForecastGroup(double[] longi...
...tude,
            double[] latitude, double centroidLongitude, double centroi...
...dLatitude,
            double forecastArea){
        ArrayList<GPoint2D<Double>> convectionBoundary =
                new ArrayList<GPoint2D<Double>>(longitude.length);
        for(int i=0; i<longitude.length; i++){
            // Check if three points in a row have the same latitude or...
... longitude
            if(convectionBoundary.size() > 2){
                GPoint2D<Double> pt1 = convectionBoundary.get(convectio...
...nBoundary.size() - 2);
                GPoint2D<Double> pt2 = convectionBoundary.get(convectio...
...nBoundary.size() - 1);

                // Check for three longitudes in a row
```

```
                if(pt1.getLongitude() == pt2.getLongitude() && pt1.getL...
...ongitude() == longitude[i]){
                    pt2.setLatitude(latitude[i]);
                    continue;
                }

                // Check for three latitudes in a row
                if(pt1.getLatitude() == pt2.getLatitude() && pt1.getLat...
...itude() == latitude[i]){
                    pt2.setLongitude(longitude[i]);
                    continue;
                }
            }

            GPoint2D<Double> pt = new GPoint2D<Double>(longitude[i], la...
...titude[i]);
            convectionBoundary.add(pt);
        }
        convectionForecastGroupLocations_.add(convectionBoundary);

        convectionForecastCentroidLocations_.add(new GPoint2D<Double>(c...
...entroidLongitude,
                centroidLatitude));

        convectionForecastArea_.add(forecastArea);

        return this;
    }

    public ConvectionForecast addConvectionForecastTops(int index, int ...
...tops, GPoint2D<Double> location){
        if(observedConvectionTops_.size() == 0){
            for(int i=0; i<noConvectionForecastGroups(); i++){
                convectionForecastTops_.add(tops);
                convectionForecastTopLocations_.add(location);
            }
        }else{
            convectionForecastTops_.set(index, tops);
            convectionForecastTopLocations_.set(index, location);
        }

        return this;
    }

    public int getConvectionForecastTops(int index){
        return convectionForecastTops_.get(index);
    }
```

```
    public GPoint2D<Double> getConvectionForecastTopsLocation(int index...
...){
        return convectionForecastTopLocations_.get(index);
    }

    public ConvectionForecast addObservedConvectionTops(int tops, GPoin...
...t2D<Double> location){
        observedConvectionTops_.add(tops);
        observedConvectionTopLocations_.add(location);

        return this;
    }

    // Number of convection groups
    public int noConvectionForecastGroups(){
        return convectionForecastGroupLocations_.size();
    }

    // Latitude of the convection locations
    public double[] getConvectionForecastGroupLatitude(int index){
        ArrayList<GPoint2D<Double>> pts = convectionForecastGroupLocati...
...ons_.get(index);

        double[] lats = new double[pts.size()];
        for(int i=0; i<pts.size(); i++){
            lats[i] = pts.get(i).getLatitude();
        }

        return lats;
    }

    // Longitude of the convection locations
    public double[] getConvectionForecastGroupLongitude(int index){
        ArrayList<GPoint2D<Double>> pts = convectionForecastGroupLocati...
...ons_.get(index);

        double[] lons = new double[pts.size()];
        for(int i=0; i<pts.size(); i++){
            lons[i] = pts.get(i).getLongitude();
        }

        return lons;
    }

    // Latitude/Longitude centroid of the convection forecast
    public GPoint2D<Double> getConvectionForecastGroupCentroid(int inde...
```

```
...x){
        return convectionForecastCentroidLocations_.get(index);
    }

    // Area (nm^2) of NCWF convection forecast
    public double getConvectionForecastArea(int index){
        return this.convectionForecastArea_.get(index);
    }

    // Add polygons of observed convection
    public ConvectionForecast addObservedConvection(double[] convection...
...Lat,
            double[] convectionLon, double[] buffer10NmLat, double[] bu...
...ffer10NmLon,
            double[] buffer20NmLat, double[] buffer20NmLon){

        for(int i=0; i<convectionLat.length; i++){
            GPoint2D<Double> convectionPt = new GPoint2D<Double>(
                    convectionLon[i],convectionLat[i]);
            observedConvectionPolygon_.add(convectionPt);
        }

        for(int i=0; i<buffer10NmLat.length; i++){
            GPoint2D<Double> buffer10NmPt = new GPoint2D<Double>(
                    buffer10NmLon[i],buffer10NmLat[i]);
            observedConvection10NmBufferPolygon_.add(buffer10NmPt);
        }

        for(int i=0; i<buffer20NmLat.length; i++){
            GPoint2D<Double> buffer20NmPt = new GPoint2D<Double>(
                    buffer20NmLon[i],buffer20NmLat[i]);
            observedConvection20NmBufferPolygon_.add(buffer20NmPt);
        }

        return this;
    }


    /** Polygon of observed convection with 10 nm buffer. */
    public ArrayList<GPoint2D<Double>> observedConvection10NmBufferPoly...
...gon(){
        return observedConvection10NmBufferPolygon_;
    }

    /** Observed convection tops */
    public ArrayList<Integer> observedConvectionTops(){
        return observedConvectionTops_;
```

```
    }

    /** Locations of the observed convection tops */
    public ArrayList<GPoint2D<Double>> observedConvectionTopLocations()...
...{
        return observedConvectionTopLocations_;
    }

}
```

ConvectionForecastDistribution.java

```
/*
 * ConvectionForecastDistribution.java
 *
 * Created on September 25, 2007, 8:44 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author Jeff
 */
import java.util.*;
public class ConvectionForecastDistribution {
    private ArrayList<Double> forecastArea_;

    private double maxForecastArea_;

    // Area bias with no buffer
    private ArrayList<Double> weibullLambda0nmBuffer_;
    private ArrayList<Double> weibullK0nmBuffer_;

    // Area bias with 10 nm buffer
    private ArrayList<Double> weibullLambda10nmBuffer_;
    private ArrayList<Double> weibullK10nmBuffer_;

    // Area bias with 20 nm buffer
    private ArrayList<Double> weibullLambda20nmBuffer_;
    private ArrayList<Double> weibullK20nmBuffer_;

    // Centroid distance parameter
    private ArrayList<Double> weibullLambdaCentroidDist_;
    private ArrayList<Double> weibullKCentroidDist_;


    /** Creates a new instance of ConvectionForecastDistribution */
    public ConvectionForecastDistribution() {
        forecastArea_ = new ArrayList<Double>(15);

        maxForecastArea_ = 0.0D;

        weibullLambda0nmBuffer_ = new ArrayList<Double>(15);
        weibullK0nmBuffer_ = new ArrayList<Double>(15);

        weibullLambda10nmBuffer_ = new ArrayList<Double>(15);
```

```
        weibullK10nmBuffer_ = new ArrayList<Double>(15);

        weibullLambda20nmBuffer_ = new ArrayList<Double>(15);
        weibullK20nmBuffer_ = new ArrayList<Double>(15);

        weibullLambdaCentroidDist_ = new ArrayList<Double>(15);
        weibullKCentroidDist_ = new ArrayList<Double>(15);
    }

    public ConvectionForecastDistribution addData(double forecastArea, ...
...double weibullLambda0nmBuffer,
            double weibullK0nmBuffer, double weibullLambda10nmBuffer, d...
...ouble weibullK10nmBuffer,
            double weibullLambda20nmBuffer, double weibullK20nmBuffer,
            double weibullLambdaCentroidDist, double weibullKCentroidDi...
...st){

        if(forecastArea > maxForecastArea_) maxForecastArea_ = forecast...
...Area;

        forecastArea_.add(forecastArea);
        weibullLambda0nmBuffer_.add(weibullLambda0nmBuffer);
        weibullK0nmBuffer_.add(weibullK0nmBuffer);
        weibullLambda10nmBuffer_.add(weibullLambda10nmBuffer);
        weibullK10nmBuffer_.add(weibullK10nmBuffer);
        weibullLambda20nmBuffer_.add(weibullLambda20nmBuffer);
        weibullK20nmBuffer_.add(weibullK20nmBuffer);
        weibullLambdaCentroidDist_.add(weibullLambdaCentroidDist);
        weibullKCentroidDist_.add(weibullKCentroidDist);

        return this;
    }

    public double getWeibullLambda0nmBuffer(double forecastArea){
        int forecastAreaRounded = (int)Math.round(forecastArea/50.0D);

        if(forecastAreaRounded >= weibullLambda0nmBuffer_.size()){
            return weibullLambda0nmBuffer_.get(weibullLambda0nmBuffer_....
...size() - 1);
        }else{
            return weibullLambda0nmBuffer_.get(forecastAreaRounded);
        }
    }

    public double getWeibullK0nmBuffer(double forecastArea){
        int forecastAreaRounded = (int)Math.round(forecastArea/50.0D);
```

```
        if(forecastAreaRounded >= weibullK0nmBuffer_.size()){
            return weibullK0nmBuffer_.get(weibullK0nmBuffer_.size() - 1...
...);
        }else{
            return weibullK0nmBuffer_.get(forecastAreaRounded);
        }
    }

    public double getWeibullLambda10nmBuffer(double forecastArea){
        int forecastAreaRounded = (int)Math.round(forecastArea/50.0D);

        if(forecastAreaRounded >= weibullLambda10nmBuffer_.size()){
            return weibullLambda10nmBuffer_.get(weibullLambda10nmBuffer...
..._.size() - 1);
        }else{
            return weibullLambda10nmBuffer_.get(forecastAreaRounded);
        }
    }

    public double getWeibullK10nmBuffer(double forecastArea){
        int forecastAreaRounded = (int)Math.round(forecastArea/50.0D);

        if(forecastAreaRounded >= weibullK10nmBuffer_.size()){
            return weibullK10nmBuffer_.get(weibullK10nmBuffer_.size() -...
... 1);
        }else{
            return weibullK10nmBuffer_.get(forecastAreaRounded);
        }
    }

    public double getWeibullLambda20nmBuffer(double forecastArea){
        int forecastAreaRounded = (int)Math.round(forecastArea/50.0D);

        if(forecastAreaRounded >= weibullLambda20nmBuffer_.size()){
            return weibullLambda20nmBuffer_.get(weibullLambda20nmBuffer...
..._.size() - 1);
        }else{
            return weibullLambda20nmBuffer_.get(forecastAreaRounded);
        }
    }

    public double getWeibullK20nmBuffer(double forecastArea){
        int forecastAreaRounded = (int)Math.round(forecastArea/50.0D);

        if(forecastAreaRounded >= weibullK0nmBuffer_.size()){
            return weibullK20nmBuffer_.get(weibullK20nmBuffer_.size() -...
... 1);
```

```
        }else{
            return weibullK20nmBuffer_.get(forecastAreaRounded);
        }
    }

    public double getWeibullLambdaCentroidDist(double forecastArea){
        int forecastAreaRounded = (int)Math.round(forecastArea/50.0D);

        if(forecastAreaRounded >= weibullLambdaCentroidDist_.size()){
            return weibullLambdaCentroidDist_.get(weibullLambdaCentroid...
...Dist_.size() - 1);
        }else{
            return weibullLambdaCentroidDist_.get(forecastAreaRounded);...
...         }
    }

    public double getWeibullKCentroidDist(double forecastArea){
        int forecastAreaRounded = (int)Math.round(forecastArea/50.0D);

        if(forecastAreaRounded >= weibullKCentroidDist_.size()){
            return weibullKCentroidDist_.get(weibullKCentroidDist_.size...
...() - 1);
        }else{
            return weibullKCentroidDist_.get(forecastAreaRounded);
        }
    }

}
```

Convolution.java

```
/*
 * Convolution.java
 *
 * Created on October 23, 2007, 6:10 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author jeff
 */
import java.util.*;
public class Convolution {
    private KernelDensity densityA_;
    private KernelDensity densityB_;

    //private KernelDensity densityResult_;

    /** Creates a new instance of Convolution */
    public Convolution(KernelDensity densityA, KernelDensity densityB){...
...        densityA_ = densityA;
        densityB_ = densityB;
    }

    public KernelDensity run(){
        // Initialize the density object to return
        ArrayList<Double> returnDensities = new ArrayList<Double>(densi...
...tyA_.noPoints());
        for(int i=0; i<densityA_.noPoints(); i++) returnDensities.add(0...
....0D);

        KernelDensity densityResult = new KernelDensity(APData.NULLSTRI...
...NG,
                densityA_.getValues(), returnDensities);

        // Get min and max values
        int maxIdx = densityResult.noPoints() - 1;
        double minValue = densityResult.getValue(0);
        double maxValue = densityResult.getValue(maxIdx);

        double convMinValue = minValue*2;
        double convMaxValue = maxValue*2;

        int resultLength = 2*densityResult.noPoints() - 1;
```

```
        double diff = (convMaxValue - convMinValue)/(resultLength - 1);...
...
        // Perform the convolution
        int idx = 0;
        for(int j=0; j<resultLength; j++){
            double curValue = j*diff + convMinValue;
            if(curValue < minValue || curValue > maxValue) continue;

            for(int k=0; k <= j; k++){
                // Check index dimensions
                if(j-k > maxIdx) continue;
                if(k > maxIdx) break;

                double tmpVal = densityResult.getDensity(idx);
                tmpVal += densityA_.getDensity(k)*densityB_.getDensity(...
...j-k);
                densityResult.setDensity(idx, tmpVal);
            }
            idx++;
        }

        // Get the normalization value
        double normVal = 0.0D;
        for(int j=0; j<maxIdx; j++) normVal += densityResult.getDensity...
...(j);


        // Normalize the result
        for(int i=0; i<maxIdx; i++){
            double tmpVal = densityResult.getDensity(i);
            tmpVal /= normVal;
            densityResult.setDensity(i, tmpVal);
        }

        return densityResult;
    }

}
```

DijkstraAdvanced.java

```
// ************************************************ //
// Dijkstra Algorithm
//
// JFL - 2002
//
// http://www.nimbustier.net/publications/djikstra/djikstra.html
//
// ************************************************ //
// Mes beaux imports
import java.util.*;
/**
 * Provides a advanced algorithm to find distance from a node
 * and the paths corresponding to this node.
 * <p>
 * After constructing the dijkstra object, two steps are necessary to o...
...btains
 * shortest paths or distances. Choose a node from where you want to co...
...mpute
 * distances and launch <code>valuateFromSource</code>. Then, you can a...
...ccess
 * the values via other access methods.
 * </p>
 * <p>
 * Nota that <code>DISTANCE_MAX</code> is a constant representing the m...
...aximum
 * distance beetween two nodes (modelize the infinite value). It can be...
... a
 * serious limitation for the use of algorithm.
 * <p>
 * The distance on edges are read with getDoubleValue with the paramete...
...r
 * DIJKSTRADISTANCE. You can change this string to another string to re...
...ad
 * another value.
 * </p>
 * <p>
 * Note that all values on edges must be stored as Double.
 * </p>
 * @see #valuateFromSource valuateFromSource
 * @author  jflaland@sophia.inria.fr
 * @version 23/05/2002
 *
 * Modified December 07 - Jeff Henderson
 */
public class DijkstraAdvanced
{
```

```
    final static double DISTANCE_MAX = Double.MAX_VALUE;
    private Network g_;
    private HashMap<Node3D<Double>, Node3D<Double>> distances_;
    private HashMap<Node3D<Double>, Link3D> predecessor_;
    private Node3D<Double> start_;
    private int indexOfLinkValue_;
    private int indexOfNodeValue_;

    private boolean stopAtDestination_;
    private Node3D<Double> destinationNode_;
    /**
     * Default constructor.
     *
     * When constructed, execute the <code>valuateFromSource</code> fun...
...ction to
     * compute the distances and path and then, get these results via o...
...ther
     * access method of the class.
     *
     * @param g the graph used by Dijkstra algorithm
     * @param nameOfValue the string name of the value to consider as d...
...istances on arcs
     *
     */
    // Constructeur
    public DijkstraAdvanced(Network g, int indexOfLinkValue, int indexO...
...fNodeValue)
    {
        g_ = g;
        distances_ = new HashMap<Node3D<Double>, Node3D<Double>>();
        indexOfLinkValue_ = indexOfLinkValue;
        indexOfNodeValue_ = indexOfNodeValue;
        stopAtDestination_ = false;
    }

    public DijkstraAdvanced(Network g, int indexOfLinkValue, int indexO...
...fNodeValue,
            Node3D<Double> destinationNode)
    {
        g_ = g;
        distances_ = new HashMap<Node3D<Double>, Node3D<Double>>();
        indexOfLinkValue_ = indexOfLinkValue;
        indexOfNodeValue_ = indexOfNodeValue;
        stopAtDestination_ = true;
        destinationNode_ = destinationNode;
    }
    public void valuateFromSource2(Node3D<Double> startNode){
```

```
        if(startNode == null) return;

        start_ = startNode;


        NodeComparator nc = new NodeComparator(indexOfNodeValue_);

        PriorityQueue<Node3D<Double>> c = new PriorityQueue<Node3D<Doub...
...le>>(
                g_.noNodes(), nc);

        ArrayList<Node3D<Double>> nodesNotNaN = g_.getNodeCollectionByN...
...anIndex(indexOfNodeValue_);
        Iterator<Node3D<Double>> itC = nodesNotNaN.iterator();

        // Initialize the predecessors
        predecessor_ = new HashMap<Node3D<Double>, Link3D>(nodesNotNaN....
...size());

        start_.setDoubleValue(indexOfNodeValue_, 0D);

        // Set predecessor of start node to null
        predecessor_.put(start_, null);

        // Add start node to the priority queue
        c.add(start_);

        // Initialize distance labels
        while (itC.hasNext())
        {
            Node3D<Double> currentNode = itC.next();

            if (!currentNode.equals(start_))
                currentNode.setDoubleValue(indexOfNodeValue_, DISTANCE_...
...MAX);


        }

        while(!c.isEmpty()){
            Node3D<Double> minNode = c.poll();

            for(Link3D outlink:minNode.getOutLinkList()){
                Node3D<Double> toNode = outlink.getToNode();

                double value = minNode.getDoubleValue(indexOfNodeValue_...
...) +
```

```
                        outlink.getDoubleValue(indexOfLinkValue_);
                double dj = toNode.getDoubleValue(indexOfNodeValue_);
                if(dj > value){
                    if(dj == DISTANCE_MAX){
                        toNode.setDoubleValue(indexOfNodeValue_, value)...
...;
                        predecessor_.put(toNode, outlink);
                        c.add(toNode);
                    }else{
                        toNode.setDoubleValue(indexOfNodeValue_, value)...
...;
                        predecessor_.put(toNode, outlink);
                    }

                    if(stopAtDestination_ && toNode.equals(destinationN...
...ode_)){
                        return;
                    }
                }

            }
        }

    }

    /**
     * Computes the shortest distances and paths from node u.
     *
     * @param u the abstractnode source from where computing distances
     *
     */
    public void valuateFromSource(Node3D<Double> startNode)
    {
        start_ = startNode;
        // Ensembles C et D
        FibonacciHeap c = new FibonacciHeap(g_,indexOfNodeValue_);
        distances_.clear();
        ArrayList<Node3D<Double>> nodesNotNaN = g_.getNodeCollectionByN...
...anIndex(indexOfNodeValue_);
        Iterator<Node3D<Double>> itC = nodesNotNaN.iterator(); //g_.get...
...NodeValueIterator();
        // Modification pour la source...
        start_.setDoubleValue(indexOfNodeValue_, 0D);
        // Pr?paration des tables
        while (itC.hasNext())
        {
            Node3D<Double> currentNode = itC.next();
```

```
            distances_.put(currentNode,start_);
            // Valeur par defaut
            if (!currentNode.equals(start_))
                currentNode.setDoubleValue(indexOfNodeValue_, DISTANCE_...
...MAX);
            ArrayList<Node3D<Double>> neighbors = start_.getNeighbors(i...
...ndexOfNodeValue_);
            if (neighbors.contains(currentNode)) {
                for (Link3D ae : start_.getEdgesTo(currentNode)) {
                    if(Double.isNaN(ae.getDoubleValue(indexOfLinkValue_...
...))) continue;
                    if (currentNode.getDoubleValue(indexOfNodeValue_) >...
...= 0.0D) {
                        if (Double.compare(currentNode.getDoubleValue(i...
...ndexOfNodeValue_),
                                ae.getDoubleValue(indexOfLinkValue_)) >...
... 0) {
                            currentNode.setDoubleValue(indexOfNodeValue...
..._,
                                    ae.getDoubleValue(indexOfLinkValue_...
...));
                        }
                    }
                }
            }

        }
        // Construction de c
        Iterator<Node3D<Double>> itNodes = nodesNotNaN.iterator(); //g_...
....getNodeValueIterator();
        while (itNodes.hasNext())
        {
            Node3D<Double> n = itNodes.next();

            if (!n.equals(start_))
            {
                int index = c.insert(n);
                n.setIntegerValue(indexOfNodeValue_, index);
            }
        }

        //ArrayList<Node3D<Double>> enteredAlready =
        //        new ArrayList<Node3D<Double>>(nodesNotNaN.size());
        while (!c.isEmpty())
        {
            Node3D<Double> minNode = null;
                //AbstractNode minNode = minDAnode.getNode();
```

```
                minNode = (Node3D<Double>)c.findMin(); // Ce sommet cha...
...nge d'ensemble
                c.deleteMin();
            // On met ? jour les nodes li?s ? minNode
            for(Link3D currentEdge2 : minNode.getOutLinkList()){
                if(Double.isNaN(currentEdge2.getDoubleValue(indexOfLink...
...Value_))) continue;

                Node3D<Double> currentNode2 = currentEdge2.getToNode();...
...
                if(Double.isNaN(currentNode2.getDoubleValue(indexOfNode...
...Value_)))
                    continue;

                //if(enteredAlready.contains(currentNode2)) continue;

                if (minNode.getDoubleValue(indexOfNodeValue_) +
                        currentEdge2.getDoubleValue(indexOfLinkValue_) ...
...<
                        currentNode2.getDoubleValue(indexOfNodeValue_))...
...               {
                    currentNode2.setDoubleValue(indexOfNodeValue_,
                            minNode.getDoubleValue(indexOfNodeValue_) +...
...                                 currentEdge2.getDoubleValue(indexOfLink...
...Value_));
                    distances_.put(currentNode2,minNode);

                    //enteredAlready.add(currentNode2);
                    try{
                    c.FibHeapDecreaseKey(c.getNode(currentNode2.getInte...
...gerValue(indexOfNodeValue_)), currentNode2);
                    }catch(NullPointerException e){
                        int errorsNull = 0;
                    }
                }
            }
        }
    }
/**
  * Returns the distance from source of a node.
  *
  * @param v the node wanted to know the distance.
  * @return the distance in an integer value.
  *
  */
  public Double getDistanceTo(Node3D<Double> v)
  {
```

```
      return v.getDoubleValue(indexOfNodeValue_);
  }
/**
  * Returns the chain form source to node v.
  *
  * @param v the node ending the shortest chain starting at source node...
....
  * @return the chain form source to node v.
  *
  */
    public ArrayList<Link3D> getShortestPathTo(Node3D<Double> v) {
        ArrayList<Link3D> path = new ArrayList<Link3D>();
        if (v.equals(start_) || v.getDoubleValue(indexOfNodeValue_) == ...
...DISTANCE_MAX) {
            return null;
        } else {
            Node3D<Double> start = v;
            while (!start.equals(start_)) {
                Node3D<Double> prec = distances_.get(start);
                Link3D leadsToLink = null;
                for(Link3D link:prec.getOutLinkList()){
                    if(link.leadsTo(start)){
                        leadsToLink = link;
                        break;
                    }
                }
                path.add(0,leadsToLink);
                start = prec;
            }
            return path;
        }
    }

    public ArrayList<Link3D> getShortestPathTo2(Node3D<Double> v){
        ArrayList<Link3D> path = new ArrayList<Link3D>();

        if (v == null || v.equals(start_) || v.getDoubleValue(indexOfNo...
...deValue_) == DISTANCE_MAX) {
            return path;
        } else {
            Node3D<Double> currNode = v;

            while(!currNode.equals(start_)){
                try {
                    Link3D inLink = predecessor_.get(currNode);
                    path.add(0,inLink);
                    currNode = inLink.getFromNode();
```

```
                } catch (NullPointerException e) {
                    return new ArrayList<Link3D>();
                }
            }

            return path;
        }
    }
/**
  * Returns the current starting node.
  *
  * @return the current starting node from where to compute..
  *
  */
  public Node3D<Double> getStartNode()
  {
    return start_;
  }

  public static ArrayList<ArrayList<Link3D>> KShortestPaths(int k,
          Node3D<Double> src, Node3D<Double> dest,
          int indexForLink, int indexForNode, Network n){

      ArrayList<ArrayList<Link3D>> kPaths = new
              ArrayList<ArrayList<Link3D>>(k);

      for(int k2=0; k2<k; k2++){
          DijkstraAdvanced spa = new DijkstraAdvanced(n,indexForLink,in...
...dexForNode,dest);
          spa.valuateFromSource2(src);
          ArrayList<Link3D> path = spa.getShortestPathTo2(dest);

          if(k2 == 0 && (path == null || path.size() == 0)){
              return null;
          }else if(k2 > 0 && (path == null || path.size() == 0)){
              kPaths.add(kPaths.get(kPaths.size() - 1));
          }else{
              kPaths.add(path);

//              ArrayList<Node3D<Double>> nodes = APData.network.getNod...
...eAirportSet();
//              int j;
//              for(j=0; j<nodes.size(); j++){
//                  Node3D<Double> node = nodes.get(j);
//                  if(Double.isNaN(node.getDoubleValue(indexForNode))...
...{
//                      node.setDoubleValue(indexForNode, Double.MIN_VA...
```

```
...LUE);
//                  }
//              }

              int j;
              for(j=1; j<path.size()-2; j++){
                  Link3D link = path.get(j);

                  // Set node labels to invalid
                  Node3D<Double> toNode = link.getToNode();
                  toNode.setDoubleValue(indexForNode, Double.NaN);

                  // Only need to set one label to invalid
                  break;

                  // Alternatively, could set the link length to a larg...
...e value
                  //link.setDoubleValue(indexForLink, Double.MAX_VALUE)...
...;
              }
          }
      }


      return kPaths;
  }


}
```

EAircraft.java

```
//
//  EAircraft.java
//  AirPlanJ
//
//  Created by jeff on 8/4/06.
//  Copyright 2006. All rights reserved.
//
import java.util.*;
public class EAircraft {
    /** Typical number of flight levels. */
    static final int NUM_FLIGHT_LEVELS = 25;

    /* Pre-allocate based on distinct number of aircraft types */
    static final int INIT_NUM_AIRCRAFT = 400;

    /** Aircraft identifier (callsign) */
    private String callSign_;

    /** Aircraft alias number */
    private int aliasNumber_;

    /** Cruise TAS [kts] */
    private HashMap<Double, Double> cruiseAirspeed_;

    /** Climb TAS [kts] */
    private HashMap<Double, Double> climbAirspeed_;

    /** Descent TAS [kts] */
    private HashMap<Double, Double> descentAirspeed_;

    /** Rate of climb [fpm] used for altitude filter */
    private HashMap<Double, Double> rateOfClimb_;

    /** Rate of descent [fpm] used for altitude filter */
    private HashMap<Double, Double> rateOfDescent_;

    /** Distance/Time for Climb/Descent by cruising flight level */
    private HashMap<Double, Double> climbDistance_;
    private HashMap<Double, Double> climbTime_;
    private HashMap<Double, Double> descentDistance_;
    private HashMap<Double, Double> descentTime_;

    /** Pre-processing of climb and descent portions of flight
     * Time is in minutes.
     */
    class Preprocessed {
```

```
        public double altitude, time, distance;
        public Preprocessed(double altitude_, double time_, double dist...
...ance_) {
            altitude = altitude_;
            time = time_;
            distance = distance_;
        }
    }
    private ArrayList<Preprocessed> preprocessedClimb_;
    private ArrayList<Preprocessed> preprocessedDescent_;

    private double[][] preprocessedClimbArray_;
    private double[][] preprocessedDescentArray_;

    /** Aircraft using alias */
    private static HashMap<Integer, ArrayList<String>> aliasNumberSet_ ...
...=
            new HashMap<Integer, ArrayList<String>>(
            Settings.hashMapDefaultInitialCapacity,
            Settings.hashMapLoadFactor);

    /** Aircraft rate of climb performance limitation */
    private static HashMap<Double, Double> maxRateOfClimb_ =
            new HashMap<Double, Double>(NUM_FLIGHT_LEVELS,
            Settings.hashMapLoadFactor);

    /** Aircraft rate of descent performance limitation */
    private static HashMap<Double, Double> maxRateOfDescent_ =
            new HashMap<Double, Double>(NUM_FLIGHT_LEVELS,
            Settings.hashMapLoadFactor);


    /** Key/Value storage of aircraft based on callsign */
    private static HashMap<String, EAircraft> aircraft_ =
            new HashMap<String, EAircraft>(INIT_NUM_AIRCRAFT,
            Settings.hashMapLoadFactor);


    // Creation of an UNKNOWN aircraft as the default constructor
    public EAircraft(){
        callSign_ = APData.UNKNOWN;
        this.initializeCollections();

        this.addRateOfClimb(0.0D, -9999); // Need appropriate value for...
... altitude filter
        this.addRateOfDescent(0.0D, -9999); // Need appropriate value f...
...or altitude filter
```

```
        this.addCruiseAirspeed(0.0D, -9999);
        this.addClimbAirspeed(0.0D, -9999);
        this.addDescentAirspeed(0.0D, -9999);
    }

    public EAircraft(String callSign){
        callSign_ = callSign;

        this.initializeCollections();
    }

    public EAircraft(String callSign, int aliasNumber){
        callSign_ = callSign;
        aliasNumber_ = aliasNumber;

        // Update aliasNumberSet_
        ArrayList<String> a = aliasNumberSet_.containsKey(aliasNumber_)...
... ?
            aliasNumberSet_.get(aliasNumber_) : new ArrayList<String>()...
...;
        a.add(callSign_);
        aliasNumberSet_.put(aliasNumber_, a);

        this.initializeCollections();
    }

    private EAircraft initializeCollections(){
        cruiseAirspeed_ = new HashMap<Double, Double>(NUM_FLIGHT_LEVELS...
...,
                Settings.hashMapLoadFactor);
        climbAirspeed_ = new HashMap<Double, Double>(NUM_FLIGHT_LEVELS,...
...                 Settings.hashMapLoadFactor);
        descentAirspeed_ = new HashMap<Double, Double>(NUM_FLIGHT_LEVEL...
...S,
                Settings.hashMapLoadFactor);
        rateOfClimb_ = new HashMap<Double, Double>(NUM_FLIGHT_LEVELS,
                Settings.hashMapLoadFactor);
        rateOfDescent_ = new HashMap<Double, Double>(NUM_FLIGHT_LEVELS,...
...                 Settings.hashMapLoadFactor);
        climbDistance_ = new HashMap<Double, Double>(NUM_FLIGHT_LEVELS,...
...                 Settings.hashMapLoadFactor);
        climbTime_ = new HashMap<Double, Double>(NUM_FLIGHT_LEVELS,
                Settings.hashMapLoadFactor);
        descentDistance_ = new HashMap<Double, Double>(NUM_FLIGHT_LEVEL...
...S,
                Settings.hashMapLoadFactor);
        descentTime_ = new HashMap<Double, Double>(NUM_FLIGHT_LEVELS,
```

```
                Settings.hashMapLoadFactor);

        preprocessedClimb_ = new ArrayList<Preprocessed>(NUM_FLIGHT_LEV...
...ELS);
        preprocessedDescent_ = new ArrayList<Preprocessed>(NUM_FLIGHT_L...
...EVELS);

        return this;
    }

    /** Get aircraft callsign */
    public String getCallSign(){
        return callSign_;
    }

    public EAircraft addCruiseAirspeed(double altitude, double cruiseAi...
...rspeed){
        cruiseAirspeed_.put(altitude, cruiseAirspeed);
        return this;
    }

    /** Aircraft cruising airspeed at altitude [knots] */
    public double getCruiseAirspeed(double altitude){
        return EAircraft.returnValueAtAltitude(cruiseAirspeed_, altitud...
...e);
    }

    /** Aircraft cruising airspeed at altitude [nm/min] */
    public double getCruiseAirspeedNmPerMin(double altitude){
        double cruiseAirspeedKnots = getCruiseAirspeed(altitude);

        return cruiseAirspeedKnots / 60.0D;
    }

    /** Inverse of aircraft cruising airspeed at altitude [min/nm] */
    public double getCruiseAirspeedMinPerNm(double altitude){
        double cruiseAirspeedKnots = getCruiseAirspeed(altitude);

        return 60.0D / cruiseAirspeedKnots;
    }

    /** Average (Altitude change)/(Distance along ground) during climb ...
...*/
    public double getAverageClimbAltRoc(double altitude){
        return altitude / getClimbDistance(altitude);
    }
```

```
    /** Average (Delta time)/(Distance along ground) during climb */
    public double getAverageClimbTimeRoc(double altitude){
        return getClimbTime(altitude)/getClimbDistance(altitude);
    }

    /** Average (Altitude change)/(Distance along ground) during descen...
...t */
    public double getAverageDescentAltRoc(double altitude){
        return -1.0D * altitude / getDescentDistance(altitude);
    }

    /** Average (Delta time)/(Distance along ground) during descent */
    public double getAverageDescentTimeRoc(double altitude){
        return getDescentTime(altitude) / getDescentDistance(altitude);...
...    }

    public EAircraft addClimbAirspeed(double altitude, double climbAirs...
...peed){
        climbAirspeed_.put(altitude, climbAirspeed);
        return this;
    }

    public double getClimbAirspeed(double altitude){
        return EAircraft.returnValueAtAltitude(climbAirspeed_, altitude...
...);
    }

    public EAircraft addDescentAirspeed(double altitude, double descent...
...Airspeed){
        descentAirspeed_.put(altitude, descentAirspeed);
        return this;
    }

    public double getDescentAirspeed(double altitude){
        return EAircraft.returnValueAtAltitude(descentAirspeed_, altitu...
...de);
    }

    public EAircraft addRateOfClimb(double altitude, double rateOfClimb...
...){
        // Make sure flightLevel is an integer
        altitude = Math.rint(altitude);

        rateOfClimb_.put(altitude, rateOfClimb);

        if(maxRateOfClimb_.containsKey(altitude)){
            if(rateOfClimb > maxRateOfClimb_.get(altitude)){
```

```
                maxRateOfClimb_.put(altitude, rateOfClimb);
            }
        } else{
            maxRateOfClimb_.put(altitude, rateOfClimb);
        }

        return this;
    }

    public double getRateOfClimb(double altitude){
        return EAircraft.returnValueAtAltitude(rateOfClimb_, altitude);...
...    }

    public EAircraft addRateOfDescent(double altitude, double rateOfDes...
...cent){
        // Make sure flightLevel is an integer
        altitude = Math.rint(altitude);

        rateOfDescent_.put(altitude, rateOfDescent);

        if(maxRateOfDescent_.containsKey(altitude)){
            if(rateOfDescent > maxRateOfDescent_.get(altitude)){
                maxRateOfDescent_.put(altitude, rateOfDescent);
            }
        } else{
            maxRateOfDescent_.put(altitude, rateOfDescent);
        }

        return this;
    }

    public double getRateOfDescent(double altitude){
        return EAircraft.returnValueAtAltitude(rateOfDescent_, altitude...
...);
    }

    public static <T1> T1 returnValueAtAltitude(HashMap<Double,T1> h, d...
...ouble altitude){
        // Altitude rounded to nearest 1000 ft
        double altitude2 = Math.rint(altitude - altitude%10);

        while(!h.containsKey(altitude2)){
            altitude2 = altitude2 - 10.0D;
            if(altitude2 < 0.0D){
                altitude2 = 0.0D;
                break;
            }
```

```
        }
        return h.get(altitude2);
    }

    /** Get alias number derived from BADA & RAMS input file */
    public int getAliasNumber(){
        return aliasNumber_;
    }

    /** Get maximum rate of climb at altitude */
    public static double getMaxRateOfClimb(double altitude){
        return EAircraft.returnValueAtAltitude(maxRateOfClimb_, altitud...
...e);
    }

    /** Get maximum rate of descent at altitude */
    public static double getMaxRateOfDescent(double altitude){
        return EAircraft.returnValueAtAltitude(maxRateOfDescent_, altit...
...ude);
    }

    /** Get list of aircraft using alias number */
    public static ArrayList<String> aircraftAliasList(int aliasNumber){...
...        return aliasNumberSet_.get(aliasNumber);
    }

    /* Add an aircraft type */
    public static void addAircraftType(EAircraft a){
        aircraft_.put(a.getCallSign(), a);
    }

    /* Return an aircraft EAircraft object */
    public static EAircraft getAircraftType(String s){
        return aircraft_.get(s);
    }

    /* Does aircraft type exist? */
    public static boolean aircraftTypeExists(String s){
        if(aircraft_.containsKey(s)){
            EAircraft ac = aircraft_.get(s);
            HashMap<Double,Double> cruiseAirspeed = ac.cruiseAirspeed_;...
...            if(cruiseAirspeed.size() == 0){
                return false;
            }else{
                return true;
            }
```

```
        }else{
            return false;
        }
    }

    /* Get aircraft type keys to enumerate through hashmap */
    public static Iterator<String> getAircraftTypeIterator(){
        return aircraft_.keySet().iterator();
    }
    public EAircraft addPreprocessedClimb(double altitude, double time,...
... double distance){
        // Check that the previous point is not a duplicate
        if(preprocessedClimb_.size() > 0){
            Preprocessed pPrevious = preprocessedClimb_.get(preprocesse...
...dClimb_.size()-1);
            if(pPrevious.altitude == altitude) return this;
        }

        Preprocessed p = new Preprocessed(altitude, time, distance);
        preprocessedClimb_.add(p);

        return this;
    }

    public EAircraft addPreprocessedDescent(double altitude, double tim...
...e, double distance){
        // Check that the previous point is not a duplicate
        if(preprocessedDescent_.size() > 0){
            Preprocessed pPrevious = preprocessedDescent_.get(preproces...
...sedDescent_.size()-1);
            if(pPrevious.altitude == altitude) return this;
        }

        Preprocessed p = new Preprocessed(altitude, time, distance);
        preprocessedDescent_.add(p);

        return this;
    }
    /** Prepare climb array for MATLAB
     *  Fields: Distance, Altitude, Time
     */
    public EAircraft calculatePreprocessedClimbArray(){
        int noPoints = preprocessedClimb_.size();
        preprocessedClimbArray_ = new double[noPoints][3];

        for(int i=0; i<noPoints; i++){
            Preprocessed p = preprocessedClimb_.get(i);
```

```
            preprocessedClimbArray_[i][0] = p.distance;
            preprocessedClimbArray_[i][1] = p.altitude;
            preprocessedClimbArray_[i][2] = p.time;
        }

        return this;
    }

    public double[][] getPreprocessedClimbArray(){
        return preprocessedClimbArray_;
    }

    /** Prepare descent array for MATLAB
     *  Fields: Distance, Altitude, Time
     */
    public EAircraft calculatePreprocessedDescentArray(){
        int noPoints = preprocessedDescent_.size();
        preprocessedDescentArray_ = new double[noPoints][3];

        for(int i=0; i<noPoints; i++){
            Preprocessed p = preprocessedDescent_.get(i);
            preprocessedDescentArray_[i][0] = p.distance;
            preprocessedDescentArray_[i][1] = p.altitude;
            preprocessedDescentArray_[i][2] = p.time;
        }

        return this;
    }

    public double[][] getPreprocessedDescentArray(){
        return preprocessedDescentArray_;
    }

    public static void trimArrayLists(){
        Iterator<ArrayList<String>> e = aliasNumberSet_.values().iterat...
...or();
        while(e.hasNext()){
            ArrayList<String> al = e.next();
            al.trimToSize();
        }

        Iterator<EAircraft> e2 = aircraft_.values().iterator();
        while(e2.hasNext()){
            EAircraft ac = e2.next();
            ac.preprocessedClimb_.trimToSize();
            ac.preprocessedDescent_.trimToSize();
        }
```

```
    }

    public EAircraft addClimbDistance(double flightLevel, double distan...
...ce){
        climbDistance_.put(flightLevel, distance);
        return this;
    }

    private double getClimbDescentValue(double flightLevel, HashMap<Dou...
...ble, Double> map){
        if(map.containsKey(flightLevel)){
            return map.get(flightLevel);
        }else{
            flightLevel = flightLevel - flightLevel % 10.0D;
            if(map.containsKey(flightLevel)){
                return map.get(flightLevel);
            }else{
                return -1.0D;
            }
        }
    }

    public double getClimbDistance(double flightLevel){
        return getClimbDescentValue(flightLevel, climbDistance_);
    }

    public EAircraft addClimbTime(double flightLevel, double time){
        climbTime_.put(flightLevel, time);
        return this;
    }

    public double getClimbTime(double flightLevel){
        return getClimbDescentValue(flightLevel, climbTime_);
    }

    public EAircraft addDescentDistance(double flightLevel, double dist...
...ance){
        descentDistance_.put(flightLevel, distance);
        return this;
    }

    public double getDescentDistance(double flightLevel){
        return getClimbDescentValue(flightLevel, descentDistance_);
    }

    public EAircraft addDescentTime(double flightLevel, double time){
        descentTime_.put(flightLevel, time);
```

```
        return this;
    }

    public double getDescentTime(double flightLevel){
        return getClimbDescentValue(flightLevel, descentTime_);
    }

    // Return to here
}
```

ExternalMatlab.java

```
///*
// * ExternalMatlab.java
// *
// * Created on June 2, 2007, 9:26 PM
// *
// * To change this template, choose Tools | Template Manager
// * and open the template in the editor.
// */
//
////package AirPlanJ;
//
///**
// *
// * @author Jeff
// */
//public class ExternalMatlab {
//
//    /*
//     * Native methods corresponding to the functions
//     * that we need to call in the compiler-generated
//     * shared library.
//     */
//    public static native void applicationInitialize();
//    public static native void libInitialize();
//    public static native double[][] addMatrix(double[][] a, double[][...
...] b);
//    public static native double[][] multiplyMatrix(double[][] a, doub...
...le[][] b);
//    public static native double[][] eigMatrix(double[][] a);
//    public static native int[][] imReadGif(String f);
//    public static native void libTerminate();
//    public static native void applicationTerminate();
//
//
//    /** Creates a new instance of ExternalMatlab */
//    public ExternalMatlab() {
//    }
//
//    public static void Test(){
//        /* Input arrays */
//        double[][] a = {{1,4,7},{2,5,8},{3,6,9}};
//        double[][] b = {{1,4,7},{2,5,8},{3,6,9}};
//        /* Output array */
//        double[][] c = null;
//
//        /* Call the library functions */
```

```
//        c = addMatrix(a, b);
//        System.out.println("The value of added matrix is:");
//        display(c);
//        c = multiplyMatrix(a, b);
//        System.out.println("The value of the multiplied matrix is:");...
...//        display(c);
//        c = eigMatrix(a);
//        System.out.println("The Eigen value of the first matrix is:")...
...;
//        display(c);
//    }
//
//    /* Helper function to print out a double matrix */
//    public static void display(double[][] a) {
//        for (int i=0; i<a.length; i++) {
//            double[] row = a[i];
//            for (int j=0; j<row.length; j++) {
//                System.out.print(row[j] + " ");
//            }
//            System.out.print("\n");
//        }
//        System.out.print("\n");
//    }
//
//    public static void Test2(){
//        int[][] cdata = imReadGif(Settings.ncwfFile);
//
//        FAPioWeatherReader.ncwfImageAnalysis(cdata);
//    }
//
//    public static void Initialize(){
//        /*
//         * Load the jnimatlabdriver library
//         * containing the native method implementations.
//         */
//        String s = System.getProperty("java.class.path");
//        String[] s2 = s.split(":");
//        for(int i=0; i<s2.length; i++){
//            System.out.println(s2[i]);
//        }
//        //System.out.println(System.getProperty("java.class.path"));
//        try{
//        System.load(Settings.dependentLibFile);
//
//        }catch(UnsatisfiedLinkError e){
//            System.out.println(e.getMessage());
//            System.out.println(e.getLocalizedMessage());
```

```
//            System.out.println(e.toString());
//
//        }
//        System.load(Settings.libFile);
//
//        /* Call the application intialization routine. */
//        applicationInitialize();
//        /* Call the library intialization routine. */
//        libInitialize();
//    }
//
//    public static void Terminate(){
//        /* Call the library termination routine. */
//        libTerminate();
//        /* Call the application termination routine. */
//        applicationTerminate();
//    }
//
//}
```

FAPio.java

```
/*
 * APio.java
 *
 * Created on July 25, 2006, 12:27 AM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
/**
 *
 * @author jeff
 *
 * AirPlan file input/output methods
 */
import java.util.*;
import java.io.*;
import java.lang.*;
import java.text.*;
import java.lang.reflect.*;
public class FAPio {

    /** Three letter identifiers to filter centers for writing RAMS inp...
...ut files */
    public static final String[] writeCenters = {"ZAU", "ZID", "ZTL", "...
...ZJX", "ZMA", "ZDC", "ZOB", "ZNY", "ZBW"};
    //public static final String[] writeCenters = {"ZDC"};

    /** Sectors below minimum altitude (FL) not included in analysis */...
...    static final int minAltitude = 0;

    /** One of the date time formats used in the Seagull ETMS database ...
...*/
    static final String etmsDateFormat = "yyyy-mm-dd hh:mm:ss";

    /** Creates a new instance of FAPio */
    public FAPio() {
    }

    // Thread class to read sectors
    private static class readSectors implements Runnable{
        public void run(){
            FAPioAirspaceReader asr = new FAPioAirspaceReader(APData.ai...
...rspace);
            FAPioAirspaceWriter asw = new FAPioAirspaceWriter(APData.ai...
...rspace);
```

```
            String curFile = null;
            try{
                curFile = Settings.sectorFileName;
                asr.readSectors(Settings.sectorFileName);
                asw.writeSectorsCopy(Settings.sectorFileName);
                asr.readSectorMaps(Settings.sectorMapFileName);
                asr = null;
                asw = null;
            }catch(IOException e){
                System.out.println("Could not open file: " + curFile);
                return;
            }

        }

    }

    // Thread class to read airports
    private static class readAirports implements Runnable{
        public void run(){
            FAPioAirspaceReader asr = new FAPioAirspaceReader(APData.ai...
...rspace);
            FAPioAirspaceWriter asw = new FAPioAirspaceWriter(APData.ai...
...rspace);

            String curFile = null;
            try{
                curFile = Settings.airportFileName;
                asr.readAirports(Settings.airportFileName);
                asr.readAirportTypes(Settings.airportTypeFileName);
                asw.writeAirportsCopy(Settings.airportFileName);
                asr = null;
                asw = null;
            }catch(IOException e){
                System.out.println("Could not open file: " + curFile);
                return;
            }
        }
    }

    // Thread class to read aircraft
    private static class readAircraft implements Runnable{
        public void run(){
            FAPioAircraftReader ar = new FAPioAircraftReader();
            FAPioAircraftWriter aw = new FAPioAircraftWriter();

            String curFile = null;
```

```
            try{
                curFile = Settings.aircraftTypeFileName;
                ar.readAircraftTypes(Settings.aircraftTypeFileName);
                ar.readBada();
                curFile = Settings.aircraftPreprocessPerf;
                ar.readPreprocessedClimbAndDescent(Settings.aircraftPre...
...processPerf);
                aw.writeAircraftTypesCopy(Settings.aircraftTypeFileName...
...);
                ar = null;
                aw = null;
            }catch(IOException e){
                System.out.println("Could not open file: " + curFile);
                return;
            }

        }

    }

    // Thread class to read Navaids
    private static class readNavaids implements Runnable{
        public void run(){
            FAPioAirspaceReader asr = new FAPioAirspaceReader(APData.ai...
...rspace);
            FAPioAirspaceWriter asw = new FAPioAirspaceWriter(APData.ai...
...rspace);

            String curFile = null;
            try{
                curFile = Settings.navaidFileName;
                asr.readNavaids(Settings.navaidFileName);
                asw.writeNavaidsCopy(Settings.navaidFileName);
            }catch(IOException e){
                System.out.println("Could not open file: " + curFile);
                return;
            }
        }
    }

    // Thread class to read Fixes
    private static class readFixes implements Runnable{
        public void run(){
            FAPioAirspaceReader asr = new FAPioAirspaceReader(APData.ai...
...rspace);
            FAPioAirspaceWriter asw = new FAPioAirspaceWriter(APData.ai...
...rspace);
```

```
            String curFile = null;
            try{
                curFile = Settings.fixFileName;
                asr.readFixes(Settings.fixFileName);
                asw.writeFixesCopy(Settings.fixFileName);
            }catch(IOException e){
                System.out.println("Could not open file: " + curFile);
                return;
            }

        }
    }

    // Thread class to read Wind
    private static class readWind implements Runnable{
        public void run(){
            try{
                FAPioWindReader wr = new FAPioWindReader(APData.wind, S...
...ettings.windStations,
                        Settings.windData);
                wr.readWindStations();
                wr.readWindData();
                wr.readClosestWindStations();
                FAPioWindWriter ww = new FAPioWindWriter(APData.wind, S...
...ettings.windDirectory);
                //ww.writeWindData();   // Not required for now
                ww.writeWindStations();
            }catch(IOException e){
                System.out.println("Could not open wind file.");
                return;
            }
        }
    }

    // Thread class to read Airways
    private static class readAirways implements Runnable{
        public void run(){
            FAPioAirspaceReader asr = new FAPioAirspaceReader(APData.ai...
...rspace);
            FAPioAirspaceWriter asw = new FAPioAirspaceWriter(APData.ai...
...rspace);

            String curFile = null;
            try{
                /** International airways */
                curFile = Settings.internationalAwyFile;
```

```
                asr.readInternationalAirways(Settings.internationalAwyF...
...ile);

                /** Airways filename - not used, see International airw...
...ays */
                curFile = Settings.airwayFileName;
                asr.readAirways(Settings.airwayFileName);
                asw.writeAirwaysCopy(Settings.airwayFileName);
            }catch(IOException e){
                System.out.println("Could not open file: " + curFile);
                return;
            }
        }

    }

    // Thread class to read Navaids and Fixes in sector
    private static class readNavaidsAndFixesBySector implements Runnabl...
...e{
        public void run(){
            FAPioAirspaceReader asr = new FAPioAirspaceReader(APData.ai...
...rspace);

            String curFile = null;
            try{
                curFile = Settings.navaidSectorFileName;
                asr.readNavaidsAndFixesInSector(Settings.navaidSectorFi...
...leName,
                        Settings.fixSectorFileName);
            }catch(IOException e){
                System.out.println("Could not open file: " + curFile);
                return;
            }
        }
    }

    // Thread class to read Airway intersections with Sector boundaries...
...    private static class readAirwayIntersection implements Runnable{
        public void run(){
            FAPioAirspaceReader asr = new FAPioAirspaceReader(APData.ai...
...rspace);

            String curFile = null;
            try{
                curFile = Settings.airwaySectorIntersectionFileName;
                asr.readAirwaySectorIntersections(Settings.airwaySector...
...IntersectionFileName);
```

```
            }catch(IOException e){
                System.out.println("Could not open file: " + curFile);
                return;
            }
        }
    }

    // Thread class to read National fix file
    private static class readNatfix implements Runnable{
        public void run(){
            FAPioAirspaceReader asr = new FAPioAirspaceReader(APData.ai...
...rspace);

            String curFile = null;
            try{
                curFile = Settings.natfixFile;
                asr.readNatfix(curFile);
            }catch(IOException e){
                System.out.println("Could not open file: " + curFile);
                return;
            }
        }
    }

    // Thread class to read International waypoints
    private static class readWaypoints implements Runnable{
        public void run(){
            FAPioAirspaceReader asr = new FAPioAirspaceReader(APData.ai...
...rspace);

            String curFile = null;
            try{
                curFile = Settings.internationalWaypointsFile;
                asr.readInternationalWaypoints(curFile);
            }catch(IOException e){
                System.out.println("Could not open file: " + curFile);
                return;
            }
        }
    }

    // Thread class to read Stars
    private static class readStars implements Runnable{
        public void run(){
            FAPioAirspaceReader asr = new FAPioAirspaceReader(APData.ai...
...rspace);
```

```
            String curFile = null;
            try{
                curFile = Settings.starFile;
                asr.readStars(curFile);
            }catch(IOException e){
                System.out.println("Could not open file: " + curFile);
                return;
            }
        }
    }

    private static class readFlightPlan implements Runnable{
        String curFile = null;
        public void run(){
            try{
                curFile = Settings.etmsFlightPlanFile;
                FAPio.readFlightPlanRouteFromFlightExplorer(Settings.et...
...msFlightPlanFile);
                FAPio.readFlightPlanDataFromFlightExplorer(Settings.etm...
...sFlightPlanFile,true);
            }catch(IOException e){
                System.out.println("Could not open file: " + curFile);
                return;
            }
        }
    }

    private static class readKernelDensity implements Runnable{
        String curFile = null;
        public void run(){
            try{
                curFile = Settings.kernelDensityFileName;
                KernelReader kr = new KernelReader();
                kr.readKernelDensities(curFile);
            }catch(IOException e){
                System.out.println("Could not open file: " + curFile);
                return;
            }
        }
    }



    public static void readData(){

            //FAPioAirspaceReader asr = new FAPioAirspaceReader(APData....
...airspace);
```

```
            //FAPioAirspaceWriter asw = new FAPioAirspaceWriter(APData....
...airspace);

            /** NAVAIDs filename */
            Thread tNavaid = new Thread(new readNavaids());
            tNavaid.start();

            /** Fixes filename */
            Thread tFix = new Thread(new readFixes());
            tFix.start();

            /** Airport filename */
            Thread tAirport = new Thread(new readAirports());
            tAirport.start();

            /** Sector filename */
            Thread tSector = new Thread(new readSectors());
            tSector.start();

            /** Aircraft types filename */
            Thread tAircraft = new Thread(new readAircraft());
            tAircraft.start();

            /** Wind data */
            Thread tWind = new Thread(new readWind());
            tWind.start();

            try{
                tAirport.join();
                tNavaid.join();
                tFix.join();
            }catch(InterruptedException e){
                System.out.println("Read of data interrupted!");
            }

            /** Airway reader */
            Thread tAirway = new Thread(new readAirways());
            tAirway.start();

            try{
                tSector.join();
                tAircraft.join();
                tWind.join();
            }catch(InterruptedException e){
                System.out.println("Read of data interrupted!");
            }
```

```
            /** Read location of navaids and fixes in airspace sectors ...
...*/
            Thread tNavSec = new Thread(new readNavaidsAndFixesBySector...
...());
            tNavSec.start();
            try{
                tAirway.join();
                tNavSec.join();
            }catch(InterruptedException e){
                System.out.println("Read of data interrupted!");
            }

            /** Read locations where airways intersect with sector boun...
...daries */
            Thread tIntersect = new Thread(new readAirwayIntersection()...
...);
            tIntersect.start();

            /** National fix file */
            Thread tNatfix = new Thread(new readNatfix());
            tNatfix.start();

            /** STARs filename */
            Thread tStars = new Thread(new readStars());
            tStars.start();

            try{
                tIntersect.join();
                tNatfix.join();
                tStars.join();
            }catch(InterruptedException e){
                System.out.println("Read of data interrupted!");
            }

            /** Read international waypoints */
            Thread tWaypoints = new Thread(new readWaypoints());
            tWaypoints.start();
            try{
                tWaypoints.join();
            }catch(InterruptedException e){
                System.out.println("Read of data interrupted!");
            }

            /** Read kernel densities */
            String curFile = "";
            try{
                curFile = Settings.kernelDensityFileName;
```

```
                KernelReader kr = new KernelReader();
                kr.readKernelDensities(curFile);
            }catch(IOException e){
                System.out.println("Could not open file: " + curFile);
                return;
            }
            //Thread tDensity = new Thread(new readKernelDensity());
            //tDensity.start();

            /** ETMS Flight Plan Data */
            try{
                curFile = Settings.etmsFlightPlanFile;
                FAPio.readFlightPlanRouteFromFlightExplorer(Settings.et...
...msFlightPlanFile);
                FAPio.readFlightPlanDataFromFlightExplorer(Settings.etm...
...sFlightPlanFile,true);
            }catch(IOException e){
                System.out.println("Could not open file: " + curFile);
                return;
            }

            //Thread tFlightExplorer = new Thread(new readFlightPlan())...
...;
            //tFlightExplorer.start();

            //try{
                //tDensity.join();
            //    tFlightExplorer.join();
            //}catch(InterruptedException e){
            //    System.out.println("Reading of Flight Explorer data i...
...nterrupted!");
            //}
            // Trim data after import
            APData.airspace.trimArrayLists();
            EAircraft.trimArrayLists();
    }


    public static void writeRamsData(){
        /** Directory to write RAMS input files */
        try{
            FAPioRamsWriter rw = new FAPioRamsWriter(APData.airspace);
            rw.writeRamsInputFiles(Settings.ramsFilesDir);
        } catch (IOException e) {
            System.out.println("Unable to create RAMS input files in di...
...rectory: "
                    + Settings.ramsFilesDir);
```

```
        }



    }

    /* Read ETMS radar track information */
    public static void readRadarTrackData(){
        /** Create log to track missing/bad data */
        PrintWriter err;
        try{
            err = new PrintWriter(new BufferedWriter(new FileWriter(Set...
...tings.errLogFile)));
        } catch (IOException e) {
            System.out.println("Could not open error log file: " + Sett...
...ings.errLogFile);
            return;
        }

        String curFile = APData.NULLSTRING;

        try{
//            FAPioAircraftReader ar = new FAPioAircraftReader();
//            curFile = Settings.aircraftTypeFileName;
//
//            ar.readAircraftTypes(Settings.aircraftTypeFileName);
//            ar.readBada();

            curFile = Settings.etmsFlightPlanFile;
            FAPio.readFlightPlanDataFromFlightExplorer(Settings.etmsFli...
...ghtPlanFile,false);

            curFile = Settings.etmsRadarTrackFile;
            FAPio.readRadarTrackDataFromFlightExplorer(Settings.etmsRad...
...arTrackFile,false);

            //curFile = Settings.windData;
            //FAPio.readWindData();

            err.close();
            return;
        } catch (IOException e){
            System.out.println("Unable to read file: " + curFile);
        }

    }
```

```
    public static void writeRadarTrackData(){
        try{
            FAPio.writeRadarTracksForTimeInterpolation();

            //FAPioRamsWriter rw = new FAPioRamsWriter(APData.airspace)...
...;
            //rw.writeExternalTraffic4D();

        } catch (IOException e){
            System.out.println("Unable to write flight trajectory track...
...s.");
        }
    }

    /** Parse function to get elements. Returns true if all elements ar...
...e known, false otherwise. */
    public static boolean parseEtmsRoute(String route,
            ArrayList<String> elementIds, ArrayList<GPoint2D<Double>> l...
...ocations,
            ArrayList<String> elementClasses, boolean displayRouteError...
...s,
            double cruiseFlightLevel){
        Airspace a = APData.airspace;

        // Maximum number of unknown identifiers in the route string
        final int maxReadErrors = 2;
        int numReadErrors = 0;

        boolean completeRouteInfo = true;

        // Flag to check if should consider Victor Airways or Jet Route...
...s
        boolean lastAirwayVictor = false;
        boolean lastAirwayJetRoute = false;
        boolean useVictorAirways = cruiseFlightLevel <= 180.0D;
        boolean useJetRoutes = cruiseFlightLevel >= 180.0D;



        // Get the destination airport
        int lastSlash = route.lastIndexOf(APData.FORWARDSLASH);
        int routeLength = route.length();
        String route2 = route.lastIndexOf(APData.FORWARDSLASH) == route...
....length() - 5 ?
            route.substring(0, route.length() - 5) : route;

        // BWI..LDN.J134.COLNS.J6.HVQ.J6.YOCKY.DARBY3.SDF
```

```
        String[] origRoute = route2.split("[.]");   // Brackets needed ...
...due to regex rules.

        //
        ArrayList<String> jetRouteElements = new ArrayList<String>(orig...
...Route.length);

        for(int k=0; k<origRoute.length; k++){
            // Accounts for case of YNG/DO+10
            if(origRoute[k].contains(APData.FORWARDSLASH) && origRoute[...
...k].contains(APData.PLUS)){
                origRoute[k] = origRoute[k].split(APData.FORWARDSLASH)[...
...0];
            }

            // For unnamed lat lon such as 4200N/11100W or 1100/3400
            int slashIdx = origRoute[k].indexOf(APData.FORWARDSLASH);
            if(slashIdx > 0){ //origRoute[k].indexOf(APData.FORWARDSLAS...
...H) > 0){
                String locationId = origRoute[k];
                //String[] latLon = origRoute[k].split(APData.FORWARDSL...
...ASH);
                String latLon0 = origRoute[k].substring(0,slashIdx);
                String latLon1 = origRoute[k].substring(slashIdx+1,orig...
...Route[k].length());

                /* First get the latitude */
                double latitudeMultiplier = 1.0D;
                if(latLon0.endsWith("N")){
                    latLon0 = latLon0.replace("N","");
                }else if(latLon0.endsWith("S")){
                    latitudeMultiplier = - 1.0D;
                    latLon0 = latLon0.replace("S","");
                }

                String latitudeDegrees = latLon0.substring(0,latLon0.le...
...ngth() - 2);
                String latitudeMinutes = latLon0.substring(latLon0.leng...
...th() - 2);

                double latitude = latitudeMultiplier * (
                        Double.parseDouble(latitudeDegrees) +
                        Double.parseDouble(latitudeMinutes)/60.0D);

                /* Then get longitude */
                double longitudeMultiplier = 1.0D;
                if(latLon1.endsWith("E")){
```

```
                    latLon1 = latLon1.replace("E","");
                }else if(latLon1.endsWith("W")){
                    longitudeMultiplier = - 1.0D;
                    latLon1 = latLon1.replace("W","");
                }

                String longitudeDegrees = latLon1.substring(0, latLon1....
...length() - 2);
                String longitudeMinutes = latLon1.substring(latLon1.len...
...gth() - 2);

                double longitude = longitudeMultiplier * (
                        Double.parseDouble(longitudeDegrees) +
                        Double.parseDouble(longitudeMinutes)/60.0D);

//                latLon0 = latLon0.length() == 4 ? latLon0 + APData.N ...
...: latLon0;
//                latLon1 = latLon1.length() == 4 ? "0" + latLon1 + APD...
...ata.W : latLon1;
//
//                Double latitude = Double.parseDouble(latLon0.substrin...
...g(0,2))
//                + (1/60.0D)*Double.parseDouble(latLon0.substring(2,4)...
...);
//                latitude = latLon0.endsWith(APData.N) ? latitude : -1...
...*latitude;
//
//                try{
//                    double tst7 = Double.parseDouble(latLon1.substrin...
...g(0,3))
//                        + (1/60.0D)*Double.parseDouble(latLon1.substr...
...ing(3,5));
//                }catch(java.lang.NumberFormatException e7){
//                    System.out.println("ERROR LATLON: " + latLon1);
//                }
//
//                Double longitude = Double.parseDouble(latLon1.substri...
...ng(0,3))
//                + (1/60.0D)*Double.parseDouble(latLon1.substring(3,5)...
...);
//                longitude = latLon1.endsWith(APData.E) ? longitude : ...
...-1*longitude;
//

                GPoint2D<Double> pt = new GPoint2D<Double>(longitude, l...
...atitude);
                a.addFix(pt, locationId);
```

```
            }

            // Check if previous and next waypoint are on a jet route
            if(origRoute[k].length() == 0 && origRoute[k-1].length() > ...
...0
                    && origRoute[k+1].length() > 0){
                //j6Airway.onAirway(new ArrayList<String>(Arrays.asList...
...(route.split("[.]"))))

                if(a.onJetRoute(origRoute[k-1]) && a.onJetRoute(origRou...
...te[k+1])){
                    ArrayList<Airway> awys1 = a.jetRoutesByIdentifier(o...
...rigRoute[k-1]);
                    ArrayList<Airway> awys2 = a.jetRoutesByIdentifier(o...
...rigRoute[k+1]);

                    for(Airway awy1:awys1){
                        if(awys2.contains(awy1)){
                            origRoute[k] = awy1.getAirwayDesignator();
                            break;
                        }
                    }
                }

                if(origRoute[k].length() == 0 && a.onRnavRoute(origRout...
...e[k-1])
                    && a.onRnavRoute(origRoute[k+1])){
                    ArrayList<Airway> awys1 = a.rnavRoutesByIdentifier(...
...origRoute[k-1]);
                    ArrayList<Airway> awys2 = a.rnavRoutesByIdentifier(...
...origRoute[k+1]);

                    for(Airway awy1:awys1){
                        if(awys2.contains(awy1)){
                            origRoute[k] = awy1.getAirwayDesignator();
                            break;
                        }
                    }
                }

                if(origRoute[k].length() == 0 && a.onVictorAirway(origR...
...oute[k-1])
                    && a.onVictorAirway(origRoute[k+1])){
                    ArrayList<Airway> awys1 = a.victorAirwaysByIdentifi...
...er(origRoute[k-1]);
                    ArrayList<Airway> awys2 = a.victorAirwaysByIdentifi...
...er(origRoute[k+1]);
```

```
                    for(Airway awy1:awys1){
                        if(awys2.contains(awy1)){
                            origRoute[k] = awy1.getAirwayDesignator();
                            break;
                        }
                    }
                }

//                  ArrayList<String> wpts = new ArrayList<String>(2);
//                  wpts.add(origRoute[k-1]); wpts.add(origRoute[k+1]);
//
//                  Iterator<String> e = a.getJetRouteIterator();
//                  while(useJetRoutes && e.hasNext()){
//                      String jetRouteId = e.next();
//                      Airway jetRoute = a.getJetRoute(jetRouteId);
//                      if(jetRoute.onAirway(wpts)){
//                          origRoute[k] = jetRouteId;
//                          break;
//                      }
//                  }
//
//                  if(origRoute[k].length() == 0){
//                      Iterator<String> e2 = a.getRnavRouteIterator();
//                      while(useJetRoutes && e2.hasNext()){
//                          String rnavRouteId = e2.next();
//                          Airway rnavRoute = a.getRnavRoute(rnavRouteId...
...);
//                          if(rnavRoute.onAirway(wpts)){
//                              origRoute[k] = rnavRouteId;
//                              break;
//                          }
//                      }
//                  }
//
//                  if(origRoute[k].length() == 0 || lastAirwayVictor){
//                      Iterator<String> e3 = a.getVictorAirwayIterator()...
...;
//                      while(useVictorAirways && e3.hasNext()){
//                          String victorAirwayId = e3.next();
//                          Airway victorAirway = a.getVictorAirway(victo...
...rAirwayId);
//                          if(victorAirway.onAirway(wpts)){
//                              origRoute[k] = victorAirwayId;
//                              break;
//                          }
//                      }
```

```
//                }
            }


            if(origRoute[k].length() > 0 && !origRoute[k].startsWith(AP...
...Data.FORWARDSLASH)){
                jetRouteElements.add(origRoute[k]);

                boolean elementKnown = true;

                String prevStar = k > 0 ?
                    origRoute[k-1] + APData.PERIOD + origRoute[k] : APD...
...ata.NULLSTRING;
                String nextStar = k < origRoute.length -1 ?
                    origRoute[k] + APData.PERIOD + origRoute[k+1] : APD...
...ata.NULLSTRING;

                ArrayList<String> elementId = new ArrayList<String>();
                ArrayList<GPoint2D<Double>> location = new ArrayList<GP...
...oint2D<Double>>();
                ArrayList<String> elementClass = new ArrayList<String>(...
...);


                // Check condition where international airway and domes...
...tic
                // Navaid or Fix has the same identifier
                boolean origRouteKIsAirway = false;
                if(k > 0 &&
                        ((a.hasNavaid(origRoute[k]) && a.hasAirway(orig...
...Route[k]))
                        || (a.hasFix(origRoute[k]) && a.hasAirway(origR...
...oute[k])))){
                        Airway aTemp = a.getAirway(origRoute[k]);
                        if(!aTemp.onAirway(origRoute[k-1]).equals(APDat...
...a.FALSE)){
                            origRouteKIsAirway = true;
                        }
                }

                // Airport
                if((k==0 || k==origRoute.length-1) && a.airportExists(o...
...rigRoute[k])){
                    Airport apt = a.getAirport(origRoute[k]);
                    double lon = apt.getLongitude();
                    double lat = apt.getLatitude();
                    location.add(new GPoint2D<Double>(lon, lat));
```

```
                    elementId.add(origRoute[k]);
                    elementClass.add(apt.getClass().getName());
                // NAVAIDs
                }else if(!locations.isEmpty() && a.hasNavaid(origRoute[...
...k]) && !origRouteKIsAirway){
                    // Get the previous location to find the closest na...
...vaid for international flights
                    GPoint2D<Double> previousLocationPt = locations.get...
...(locations.size() - 1);
                    String closestNavaidId = a.getClosestFixNavaid(orig...
...Route[k], previousLocationPt);
                    GPoint2D<Double> ptToAdd = a.hasNavaid(closestNavai...
...dId) ?
                        a.getNavaid(closestNavaidId) : a.getFix(closest...
...NavaidId);

                    elementId.add(closestNavaidId);
                    location.add(ptToAdd);
                    elementClass.add(ptToAdd.getClass().getName());
                }else if(a.hasNavaid(origRoute[k]) && !origRouteKIsAirw...
...ay){
                    elementId.add(origRoute[k]);
                    location.add(a.getNavaid(origRoute[k]));
                    elementClass.add(a.getNavaid(origRoute[k]).getClass...
...().getName());
                    // Fixes
                }else if(!locations.isEmpty() && a.hasFix(origRoute[k])...
... && !origRouteKIsAirway){
                    // Get the previous location to find the closest fi...
...x for international flights
                    GPoint2D<Double> previousLocationPt = locations.get...
...(locations.size() - 1);
                    String closestFixId = a.getClosestFixNavaid(origRou...
...te[k], previousLocationPt);
                    GPoint2D<Double> ptToAdd = a.hasFix(closestFixId) ?...
...                        a.getFix(closestFixId) : a.getNavaid(closestFix...
...Id);

                    elementId.add(closestFixId);
                    location.add(ptToAdd);
                    elementClass.add(ptToAdd.getClass().getName());
                }else if(a.hasFix(origRoute[k]) && !origRouteKIsAirway)...
...{
                    elementId.add(origRoute[k]);
                    location.add(a.getFix(origRoute[k]));
                    elementClass.add(a.getFix(origRoute[k]).getClass()....
...getName());
```

```
                }else if (a.hasStar(prevStar)){
                    elementId.add(prevStar);
                    location.add(a.getStar(prevStar));
                    elementClass.add(a.getStar(prevStar).getClass().get...
...Name());
                }else if(a.hasStar(nextStar)){
                    elementId.add(nextStar);
                    location.add(a.getStar(nextStar));
                    elementClass.add(a.getStar(nextStar).getClass().get...
...Name());
//                }else if(a.hasFix(origRoute[k].substring(0, origRoute...
...[k].length()-1))){
//                    origRoute[k] = origRoute[k].substring(0, origRout...
...e[k].length()-1);
//                    elementId.add(origRoute[k]);
//                    location.add(a.getFix(origRoute[k]));
//                    elementClass.add(a.getFix(origRoute[k]).getClass(...
...).getName());
                    // Jet route
                }else if(k > 0 && a.hasAirway(origRoute[k])){
                    boolean nextFieldEmpty = false;

                    Airway awy = null;

                    // Use Victor Airways for low altitude only, attemp...
...t to
                    // replace with jet routes for high altitudes
                    if(elementIds.size() == 0){
                        awy = a.getAirway(origRoute[k]);
                    }else if(useVictorAirways && a.hasVictorAirway(orig...
...Route[k])){
                        awy = a.getVictorAirway(origRoute[k]);
                    }else if(a.hasJetRoute(origRoute[k])){
                        awy = a.getJetRoute(origRoute[k]);
                    }else if(a.hasRnavRoute(origRoute[k])){
                        awy = a.getRnavRoute(origRoute[k]);
                    }else if(k > 0 && useJetRoutes){// && a.hasVictorAi...
...rway(origRoute[k])
                        String previousPtId = elementIds.get(elementIds...
....size() -1);
                        String nextPtId = origRoute[k+1];

                        if(a.onJetRoute(previousPtId) && a.onJetRoute(n...
...extPtId)){
                            ArrayList<Airway> awys1 = a.jetRoutesByIden...
...tifier(previousPtId);
                            ArrayList<Airway> awys2 = a.jetRoutesByIden...
```

```
...tifier(nextPtId);

                            for(Airway awy1:awys1){
                                if(awys2.contains(awy1)){
                                    awy = awy1;
                                    break;
                                }
                            }
                        }

//                          ArrayList<String> wpts = new ArrayList<String...
...>(2);
//                          wpts.add(previousPtId); wpts.add(nextPtId);
//
//                          Iterator<String> e = a.getJetRouteIterator();...
...//                          while(useJetRoutes && e.hasNext()){
//                              String jetRouteId = e.next();
//                              Airway jetRoute = a.getJetRoute(jetRouteI...
...d);
//                              if(jetRoute.onAirway(wpts)){
//                                  awy = jetRoute;
//                                  break;
//                              }
//                          }

                        // If jet route not found use victor airway ins...
...tead
                        if(awy == null) awy = a.getVictorAirway(origRou...
...te[k]);
                    }

                    // Set default search order for airways
                    if(a.hasVictorAirway(origRoute[k])){
                        lastAirwayVictor = true;
                        lastAirwayJetRoute = false;
                    }else{
                        lastAirwayVictor = false;
                        lastAirwayJetRoute = true;
                    }

                    if(k <= origRoute.length - 3 &&
                            origRoute[k+1].length() == 0 && a.hasAirway...
...(origRoute[k+2])){
                        nextFieldEmpty = true;
                        origRoute[k+1] = awy.searchForAirwayIntersectio...
...n(a.getAirway(origRoute[k+2]));
                    }
```

```
                    // Check previous point for international designato...
...r
                    // i.e. add underscore("_") and two letter country(...
..."XX")
                    String revisedId = awy.onAirway(origRoute[k-1]);
                    if(!revisedId.equals(APData.TRUE) && !revisedId.equ...
...als(APData.FALSE) &&
                            k > 1 && elementIds.size() > 0){
                        // Reset the previous point
                        origRoute[k-1] = revisedId;

                        // Change the references in the lists
                        elementIds.set(elementIds.size()-1, revisedId);...
...                        GPoint2D<Double> fix = a.getFix(revisedId);
                        locations.set(locations.size()-1,fix);
                        elementClasses.set(elementClasses.size()-1, fix...
....getClass().getName());
                    }

                    // Check next point for international designator
                    // i.e. add underscore("_") and two letter country ...
...("XX")
                    if(k == origRoute.length - 1) continue;
                    revisedId = awy.onAirway(origRoute[k+1]);
                    origRoute[k+1] = revisedId.equals(APData.TRUE) || r...
...evisedId.equals(APData.FALSE) ?
                        origRoute[k+1] : revisedId;

                    try{
                        ArrayList<String> airwayLocIds =
                                awy.getPointIdentifiers(origRoute[k-1],...
... origRoute[k+1]);
                        elementId.addAll(airwayLocIds);



                        for(int id = 0; id<airwayLocIds.size(); id++){
                            if(a.hasNavaid(airwayLocIds.get(id))){
                                location.add(a.getNavaid(airwayLocIds.g...
...et(id)));
                                elementClass.add(a.getNavaid(airwayLocI...
...ds.get(id)).getClass().getName());
                            }else if(a.hasFix(airwayLocIds.get(id))){
                                location.add(a.getFix(airwayLocIds.get(...
...id)));
                                elementClass.add(a.getFix(airwayLocIds....
```

```
...get(id)).getClass().getName());
                            }else{
                                if(displayRouteErrors){
                                    System.out.println("Could not find ...
...airway waypoint: " + airwayLocIds.get(id));
                                }
                            }
                        }
                    }catch(NullPointerException e){
                        if(displayRouteErrors){
                            System.out.print(origRoute[k-1] + " OR " + ...
...origRoute[k+1]);
                            System.out.print(" are invalid identifiers ...
...for jet route: ");
                            System.out.println(origRoute[k] + " : " + r...
...oute2);
                        }
                    }


                }else if(k < origRoute.length - 2 && a.hasJetRoute(orig...
...Route[k+1])){
                    /* Check if next element is an international jet ro...
...ute */
                    Airway intlAirway = a.getJetRoute(origRoute[k+1]);

                    /* Update identifier for index k and k+2 to ID + "_...
..." + COUNTRY */
                    for(int k2 = 0; k2 < 3; k2 = k2 + 2){
                        String revisedId = intlAirway.onAirway(origRout...
...e[k+k2]);
                        if(!revisedId.equals(APData.FALSE) && !revisedI...
...d.equals(APData.TRUE)){
                            origRoute[k+k2] = revisedId;
                            elementId.add(revisedId);
                            location.add(a.getFix(revisedId));
                            elementClass.add(a.getFix(revisedId).getCla...
...ss().getName());
                        }else{
                            elementKnown = false;
                        }
                    }

                }else{
                    // Get identifier up to the first digit (0..9)
                    char[] origRouteChars = origRoute[k].toCharArray();...
...                    int firstDigit = 0;
```

```
                    for(int c=0; c<origRouteChars.length; c++){
                        if(Character.isDigit(origRouteChars[c])){
                            firstDigit = c;
                            break;
                        }
                    }

                    String testElement = firstDigit < 3 ? origRoute[k] ...
...:
                        origRoute[k].substring(0,firstDigit);
                    origRoute[k] = testElement;
                    if(!locations.isEmpty() && a.hasNavaid(testElement)...
...){
                        // Get the previous location to find the closes...
...t navaid for international flights
                        GPoint2D<Double> previousLocationPt = locations...
....get(locations.size() - 1);
                        String closestNavaidId = a.getClosestFixNavaid(...
...testElement, previousLocationPt);
                        GPoint2D<Double> ptToAdd = a.hasNavaid(closestN...
...avaidId) ?
                            a.getNavaid(closestNavaidId) : a.getFix(clo...
...sestNavaidId);

                        elementId.add(closestNavaidId);
                        location.add(ptToAdd);
                        elementClass.add(ptToAdd.getClass().getName());...
...                    }else if(a.hasNavaid(testElement)){
                        elementId.add(testElement);
                        location.add(a.getNavaid(testElement));
                        elementClass.add(a.getNavaid(testElement).getCl...
...ass().getName());
                    }else if(!locations.isEmpty() && a.hasFix(testEleme...
...nt)){
                        // Get the previous location to find the closes...
...t fix for international flights
                        GPoint2D<Double> previousLocationPt = locations...
....get(locations.size() - 1);
                        String closestFixId = a.getClosestFixNavaid(tes...
...tElement, previousLocationPt);
                        GPoint2D<Double> ptToAdd = a.hasFix(closestFixI...
...d) ?
                            a.getFix(closestFixId) : a.getNavaid(closes...
...tFixId);

                        elementId.add(closestFixId);
                        location.add(ptToAdd);
```

```
                        elementClass.add(ptToAdd.getClass().getName());...
...                    }else if(a.hasFix(testElement)){
                        elementId.add(testElement);
                        location.add(a.getFix(testElement));
                        elementClass.add(a.getFix(testElement).getClass...
...().getName());
                    }else{
                        elementKnown = false;
                        if(displayRouteErrors){
                            System.out.println(origRoute[k] + " : " + r...
...oute2);
                        }
                    }
                }


                if(elementKnown){
                    //for(String eleId:elementId) elementIds.add(new St...
...ring(eleId));
                    elementIds.addAll(elementId);
                    locations.addAll(location);
                    elementClasses.addAll(elementClass);
                }else{
                    numReadErrors++;
                    if(numReadErrors > maxReadErrors) completeRouteInfo...
... = elementKnown;
                }
            }
        }

        // Perform some additional checks
        //findAndExpandIdentifiersOnAirways(elementIds, locations, elem...
...entClasses);

//        if(completeRouteInfo){
//            // Check that distance between waypoints are not too far
//            return checkDistanceBetweenSuccessiveWaypoints(elementIds...
..., locations,
//                    elementClasses);
//        }else{
//            // Return false
//            return completeRouteInfo;
//        }

        return completeRouteInfo;
    }
```

```
    private static boolean checkDistanceBetweenSuccessiveWaypoints(Arra...
...yList<String> elementIds,
            ArrayList<GPoint2D<Double>> locations, ArrayList<String> el...
...ementClasses){

        // Need at least two locations to perform check
        if(locations.size() < 2) return false;

        int numFarLocations = 0;
        int indexOfFarLocation = 0;

        GPoint2D<Double> previousLocation = null;
        for(int i=1; i<locations.size(); i++){

            if(i == 1) previousLocation = locations.get(i-1);
            GPoint2D<Double> currentLocation = locations.get(i);

            double latitudeDiff = Math.abs(previousLocation.getLatitude...
...() -
                    currentLocation.getLatitude());
            double longitudeDiff = Math.abs(previousLocation.getLongitu...
...de() -
                    currentLocation.getLongitude());

            if(latitudeDiff > 5.0D || longitudeDiff > 5.0D){
                System.out.println(elementIds.get(i-1) + " : " + elemen...
...tIds.get(i));

                numFarLocations++;
                indexOfFarLocation = i;
            }else{
                previousLocation = currentLocation;
            }
        }

        if(numFarLocations == 1){
            elementIds.remove(indexOfFarLocation);
            locations.remove(indexOfFarLocation);
            elementClasses.remove(indexOfFarLocation);

            return true;
        }else if(numFarLocations == 0){
            return true;
        }else{
            return false;
        }
```

```
    }

    // Checks that identifiers on airways are adjacent
    private static void findAndExpandIdentifiersOnAirways(ArrayList<Str...
...ing> elementIds,
            ArrayList<GPoint2D<Double>> locations, ArrayList<String> el...
...ementClasses){
        Airspace a = APData.airspace;

        if(elementIds.size() < 5) return;

        HashMap<Integer, ArrayList<String>> elementsToInsert =
                new HashMap<Integer, ArrayList<String>>(
                Settings.hashMapDefaultInitialCapacity,
                Settings.hashMapLoadFactor);

        // Don't consider the first and last elements which are the ori...
...gin and
        // destination airports respectively.
        int startIdx = 1;
        int endIdx = elementIds.size() - 2;

        boolean hasElementsToInsert = false;

        for(int i=startIdx+1; i<=endIdx; i++){
            String id1 = elementIds.get(i-1);
            String id2 = elementIds.get(i);

            // First seach jet routes
            Iterator<String> e = a.getJetRouteIterator();
            while(e.hasNext()){
                Airway awy = a.getJetRoute(e.next());

                if(awy.identifierOnAirway(id1) && awy.identifierOnAirwa...
...y(id2)
                && !awy.adjacentIdentifiers(id2,id2)){

                    ArrayList<String> newElements = awy.getPointIdentif...
...iers(id1, id2);

                    if(newElements.size() > 0){
                        elementsToInsert.put(i, newElements);

                        hasElementsToInsert = true;
                    }
                }
            }
```

```
            // Then search airways
            e = a.getVictorAirwayIterator();
            while(e.hasNext()){
                Airway awy = a.getVictorAirway(e.next());

                if(awy.identifierOnAirway(id1) && awy.identifierOnAirwa...
...y(id2)
                && !awy.adjacentIdentifiers(id2,id2)){

                    ArrayList<String> newElements = awy.getPointIdentif...
...iers(id1, id2);

                    if(newElements.size() > 0){
                        elementsToInsert.put(i, newElements);

                        hasElementsToInsert = true;
                    }
                }
            }
        }

        // If no changes then return
        if(!hasElementsToInsert) return;

        // Create the new elements
        ArrayList<String> newElementIds = new ArrayList<String>();
        ArrayList<GPoint2D<Double>> newLocations = new ArrayList<GPoint...
...2D<Double>>();
        ArrayList<String> newElementClasses = new ArrayList<String>();

        for(int i=0; i<elementIds.size(); i++){
            newElementIds.add(elementIds.get(i));
            newLocations.add(locations.get(i));
            newElementClasses.add(elementClasses.get(i));

            if(elementsToInsert.containsKey(i)){
                ArrayList<String> newElements = elementsToInsert.get(i)...
...;

                for(int j=0; j<newElements.size(); j++){
                    String newId = newElements.get(j);
                    GPoint2D<Double> newPt = null;
                    String newClass = null;

                    if(a.hasNavaid(newId)){
                        newPt = a.getNavaid(newId);
```

```
                        newClass = newPt.getClass().getName();
                    }else if(a.hasFix(newId)){
                        newPt = a.getFix(newId);
                        newClass = newPt.getClass().getName();
                    }else{
                        System.out.println("Error on point: " + newId);...
...                    }

                    newElementIds.add(newId);
                    newLocations.add(newPt);
                    newElementClasses.add(newClass);

                }
            }
        }

        // Set the arrays to new values
        elementIds = newElementIds;
        locations = newLocations;
        elementClasses = newElementClasses;


    }

    /** Read route information to establish demand. */
    public static void readFlightExplorerRoute(String fileName) throws ...
...IOException{
        PrintWriter outJetRouteFlights = new PrintWriter(new BufferedWr...
...iter(new FileWriter(Settings.outJetRouteFlights)));
        StringBuilder sJetRouteFlights = new StringBuilder();

        Airspace a = APData.airspace;

        BufferedReader d = new BufferedReader(new FileReader(fileName))...
...;
        String line;

        Airway j6Airway = a.getJetRoute("J6");
        int numJ6Flights = 0;
        int numJ6Flights2 = 0;
        int numJ6FlightErrors = 0;
        while((line = d.readLine()) != null){
            String[] dataFields = line.split(APData.COMMA);
            if(dataFields.length < 19) continue;

            // Check the route
            String route = dataFields[19];
```

```
            //String route2 = "";

            // Check if on J6 or a sequence of NAVAIDs/Fixes are on the...
... jet route
            if(route.contains(".J6.") ||
                    j6Airway.onAirway(new ArrayList<String>(Arrays.asLi...
...st(route.split("[.]")))){

                numJ6Flights++;
                String originAirport = dataFields[12];


                // Variables to return from function parseEtmsRoute
                ArrayList<String> elementIds = new ArrayList<String>();...
...                ArrayList<GPoint2D<Double>> locations = new ArrayList<G...
...Point2D<Double>>();
                ArrayList<String> elementClass = new ArrayList<String>(...
...);

                boolean completeRouteInfo = FAPio.parseEtmsRoute(route,...
...                           elementIds,locations,elementClass,false, 200.0D...
...);


                if(!completeRouteInfo){
                    numJ6FlightErrors++;
                }

                if(elementIds.indexOf("13788") > -1 &&
                        elementIds.indexOf("12962") > -1 &&
                        elementIds.indexOf("13788") > elementIds.indexO...
...f("12962")){
                    for(int k=0; k<elementIds.size() - 1; k++){
                        j6Airway.addAirwayDemand(elementIds.get(k),
                                elementIds.get(k+1),
                                locations.get(k),
                                locations.get(k+1),
                                1);
                    }

                    // Route + Location Identifiers
                    /*sJetRouteFlights.append(route2);
                    for(int k=0; k<elementIds.size(); k++){
                        sJetRouteFlights.append(",");
                        sJetRouteFlights.append(elementIds.get(k));
                    }
                    sJetRouteFlights.append("\n");*/
```

```
                    // Flight Index, Identifier Index, Latitude, Longit...
...ude
                    /*numJ6Flights2++;
                    for(int k=0; k < elementIds.size(); k++){
                        sJetRouteFlights.append(numJ6Flights2);
                        sJetRouteFlights.append(",");
                        sJetRouteFlights.append(k);
                        sJetRouteFlights.append(",");
                        GPoint2D<Double> loc = locations.get(k);
                        sJetRouteFlights.append(loc.getLatitude());
                        sJetRouteFlights.append(",");
                        sJetRouteFlights.append(loc.getLongitude());
                        sJetRouteFlights.append("\n");
                    }*/

                }else{
                    numJ6FlightErrors++;
                }
            }
        }

        // Demand by airway segment
        ArrayList<AirwayDemand> demandList = j6Airway.getDemandList();
        for(int k=0; k < demandList.size(); k++){
            AirwayDemand ad = demandList.get(k);

            if(ad.getFromNode().equals(ad.getToNode())){
                continue;
            }

            sJetRouteFlights.append(ad.getFromNodeLocation().getLatitud...
...e());
            sJetRouteFlights.append(APData.COMMA);
            sJetRouteFlights.append(ad.getFromNodeLocation().getLongitu...
...de());
            sJetRouteFlights.append(APData.COMMA);
            sJetRouteFlights.append(ad.getToNodeLocation().getLatitude(...
...));
            sJetRouteFlights.append(APData.COMMA);
            sJetRouteFlights.append(ad.getToNodeLocation().getLongitude...
...());
            sJetRouteFlights.append(APData.COMMA);
            sJetRouteFlights.append(ad.getDemand());
            sJetRouteFlights.append(APData.COMMA);
            sJetRouteFlights.append(j6Airway.getLocationIndex(ad.getFro...
...mNode()));
```

```
            sJetRouteFlights.append(APData.COMMA);
            sJetRouteFlights.append(j6Airway.getLocationIndex(ad.getToN...
...ode()));

            sJetRouteFlights.append(APData.SLASH_N);
        }

        d.close();
        outJetRouteFlights.print(sJetRouteFlights.toString());
        outJetRouteFlights.close();

        System.out.print("J6 Flights: "); System.out.println(numJ6Fligh...
...ts);
        System.out.print("J6 Flights with errors/unknown route: "); Sys...
...tem.out.println(numJ6FlightErrors);

    }

    /** Read flight plan information from Seagull ETMS source */
    public static void readSeagullFlights(String fileName, PrintWriter ...
...err) throws IOException{
        int counter = 1;
        /** Show system memory information */
        Runtime rt = Runtime.getRuntime();

        Airspace a = APData.airspace;

        BufferedReader d = new BufferedReader(new FileReader(fileName))...
...;

        String line;
        while((line = d.readLine()) != null){
            String[] dataFields = line.split(APData.COMMA);
            String callSign = dataFields[1]; // However cannot use dire...
...ctly since some flight have multiple legs

            String origin = dataFields[3]; // Origin airport identifier...
...            String destination = dataFields[4]; // Destination airport ...
...identifier
            if(!a.airportExists(origin) || !a.airportExists(destination...
...)){
                err.println("Could not find airport: " + origin + " or ...
..." + destination);
                err.println(line);
                err.println("Data excluded!...");
                err.println("...");
                continue;
```

```
            }
            Airport originAirport = a.getAirport(origin);
            Airport destinationAirport = a.getAirport(destination);

            int cruiseFlightLevel = Integer.parseInt(dataFields[6]);

            String acType = dataFields[2];
            if(!EAircraft.aircraftTypeExists(acType)){
                if(cruiseFlightLevel < 300){
                    acType = "BE20";     // Substitution turboprop airc...
...raft
                } else{
                    acType = "C500";     // Substitution corporate jet
                }
            }
            EAircraft aircraftType = EAircraft.getAircraftType(acType);...
...
            Flight f = new Flight(callSign, originAirport, destinationA...
...irport,
                    aircraftType, cruiseFlightLevel);

            try{
                a.addFlight(f);
            } catch (Exception e){
                System.out.print("Could not add flight: ");
                System.out.println(callSign);
            }




        /*
         * Add trajectory information to flight
         *
         *
         */

            /** Departure time */
            String depTime = dataFields[21];
            if(depTime.length() < 19){
                err.println("Invalid departure date: " + depTime);
                err.println(line);
                err.println("Data excluded!...");
                err.println("...");
                continue;
            }
```

```
            SimpleDateFormat formatter = new SimpleDateFormat("yyyy-mm-...
...dd hh:mm:ss");
            ParsePosition pos = new ParsePosition(0);
            Date dt = formatter.parse(dataFields[21], pos);
            GregorianCalendar departureTime = new GregorianCalendar();
            departureTime.setTime(dt);
            //System.out.println(cal.get(Calendar.DAY_OF_MONTH));









            /** Waypoint information */
            FTrajectory plannedTrajectory = new FTrajectory(departureTi...
...me);
            String[] sWaypoints = dataFields[14].split(APData.SPACE);
            for(int i = 0; i < Array.getLength(sWaypoints); i++){
                String[] latLon = sWaypoints[i].split(APData.FORWARDSLA...
...SH);
                if(Array.getLength(latLon) > 1){
                    double latitude = Double.parseDouble(latLon[0])/60....
...0;
                    double longitude = Double.parseDouble(latLon[1])/60...
....0;
                    GPoint4D<Double> waypoint = new GPoint4D<Double>(lo...
...ngitude, latitude);
                    //plannedTrajectory.addWaypoint(waypoint);
                }
            }
            plannedTrajectory.trimWaypoints();

            f.setPlannedTrajectory(plannedTrajectory);

            counter++;
            /** Force garbage collection if memory running low */

            if(rt.freeMemory() < 20000){
                rt.runFinalization();
                rt.gc();
                System.out.print("Iteration: = ");
                System.out.println(counter);
            }

        }
```

```
        d.close();
    }



    // Thread class to read and store route information
    private static class readFlightPlanRoute implements Runnable{
        private static final Airspace a = APData.airspace;
        private ArrayList<String> lines;

        // Constructor
        public readFlightPlanRoute(ArrayList<String> lines_){
            lines = lines_;
        }

        public void run(){
            for(String line:lines){
                String[] dataFields = line.split(APData.COMMA);
                String route = new String(dataFields[19]);

                if(route.length() == 0) continue; //return;


                ArrayList<String> elementIds = new ArrayList<String>();...
...                 ArrayList<GPoint2D<Double>> locations = new ArrayList<G...
...Point2D<Double>>();
                ArrayList<String> elementClass = new ArrayList<String>(...
...);

                boolean completeRouteInfo = false;
                completeRouteInfo = FAPio.parseEtmsRoute(route,
                        elementIds,locations,elementClass,false, 200.0D...
...);

                if(elementIds.size() == 0) continue; //return;

                String originAirportId = elementIds.get(0);

                if(a.airportExists(originAirportId)){
                    elementIds.remove(0);
                }else{
                    continue; //return;
                }

                if(elementIds.size() == 0) continue; //return;
```

```
                String destinationAirportId = null;
                try{
                    destinationAirportId = elementIds.get(elementIds.si...
...ze() - 1);
                }catch(java.lang.ArrayIndexOutOfBoundsException e2){
                    System.out.println(originAirportId);
                }

                if(a.airportExists(destinationAirportId)){
                    elementIds.remove(elementIds.size() - 1);
                }else{
                    continue; //return;
                }

                if(elementIds.size() == 0) continue; //return;

                if(!a.hasPreferredRoute(originAirportId, destinationAir...
...portId)){
                    a.addPreferredRoute(originAirportId, destinationAir...
...portId, route);
                }

                elementIds = null;
                locations = null;
                elementClass = null;
                dataFields = null;
            }
        }
    }

    /** Archive the route information for flights with invalid/missing ...
...route information. */
    public static void readFlightPlanRouteFromFlightExplorer(String fil...
...eName) throws IOException{
        Airspace a = APData.airspace;

        BufferedReader d = new BufferedReader(new FileReader(fileName))...
...;
        String line;

        int lineCount = 0;
        int maxLines = Settings.flightPlanFileLinesToRead;
        System.out.println("Reading first " + Integer.toString(maxLines...
...) +
                " lines to obtain route information from the flight pla...
...n file.");
```

```
        // Keep track of the threads
        ArrayList<Thread> threads = new ArrayList<Thread>();
        // Send 1000 lines to each thread to reduce the overhead incurr...
...ed by
        // creating too many threads.
        boolean firstRead = true;
        int lineIdx = -1;
        int linesToRead = 1000;
        ArrayList<String> lines = new ArrayList<String>(linesToRead);

        while((line = d.readLine()) != null){
            if(lineCount++ > maxLines) break;

            lineIdx++;

            if(firstRead && lineIdx < linesToRead - 1){
                // Add up to next to last element
                lines.add(line);
            }else if(lineIdx < linesToRead - 1){
                // Reuse the lines ArrayList after the first iteration
                lines.set(lineIdx, line);
            }else if(lineIdx == linesToRead - 1){
                if(firstRead){
                    lines.add(line);
                    firstRead = false;
                }else{
                    lines.set(lineIdx, line);
                }

                // Create a copy of the lines ArrayList
                ArrayList<String> lines2 = new ArrayList<String>(lines)...
...;

                Thread t = new Thread(new readFlightPlanRoute(lines2));...
...                t.start();
                threads.add(t);

                lineIdx = -1;
            }
        }

        // Send remaining lines to a new thread
        while(lines.size() > lineIdx+1){
            lines.remove(lineIdx+1);
        }
        Thread t2 = new Thread(new readFlightPlanRoute(lines));
        t2.start();
```

```
        threads.add(t2);


        try{
            for(Thread t:threads){
                t.join();
                t = null;
            }
        }catch(InterruptedException e){
            System.out.println("Reading of route data interrupted!");
        }

        line = null;

        d.close();

    }


    // Thread class to create flights
    private static class readFlightPlanData implements Runnable{
        private static final Airspace a = APData.airspace;
        private ArrayList<String> lines;
        private boolean useUniqueId;
        public static int i=0;
        public static int j=0;
        public static int totalFlightPlans = 0;
        public static int flightPlanErrors = 0;
        public static int blankRoutes = 0;
        public static HashMap<String, Integer> unknownType;

        // Constructor
        public readFlightPlanData(ArrayList<String> lines_, boolean use...
...UniqueId_){
            lines = lines_;
            useUniqueId = useUniqueId_;
        }

        public void run(){
            for(String line:lines){
                String[] dataFields = line.split(APData.COMMA);
                totalFlightPlans++;

                // Discard flights below 18,000 ft
                int altitude = 0;
                try{
                    altitude = Integer.parseInt(dataFields[18]);
```

```
//                if(altitude < 180) continue;
                }catch(NumberFormatException e){
                    flightPlanErrors++;
                    continue; //return;
                }

                // Don't consider flights that are in the filter.
                if(a.flightInFlightFilter(dataFields[6])) continue; //r...
...eturn;

                // Need to consider both callsign and CID
                a.addComputerIdentifierToCallsign(new String(dataFields...
...[6]),
                        new String(dataFields[7]));

                // Use the unique FID field if useUniqueId set
                String aircraftId = useUniqueId ? dataFields[6] + APDat...
...a.UNDERSCORE + dataFields[0] :
                    dataFields[6];


                String acType = dataFields[10];

//            if(aircraftId.startsWith("JBU255")){
//                System.out.println(line);
//            }

                if(EAircraft.aircraftTypeExists(acType)){
                    if(!a.containsFlight(aircraftId)){
                        EAircraft aircraftType = EAircraft.getAircraftT...
...ype(acType);

                        // Variables to return from function parseEtmsR...
...oute
                        Airport originAirport = null;
                        Airport destinationAirport = null;
                        ArrayList<String> elementIds = new ArrayList<St...
...ring>();
                        ArrayList<GPoint2D<Double>> locations = new Arr...
...ayList<GPoint2D<Double>>();
                        ArrayList<String> elementClass = new ArrayList<...
...String>();
                        String route = dataFields[19];

                        boolean completeRouteInfo = false;
                        completeRouteInfo = FAPio.parseEtmsRoute(route,...
...                                 elementIds,locations,elementClass,false...
```

```
..., altitude);

                        // Check for enough elements to get origin and ...
...destination airport
                        if(elementIds.size() < 2){
                            flightPlanErrors++;
                            continue; //return;
                        }

                        // Get the origin airport
                        if(a.airportExists(dataFields[12])){
                            originAirport = a.getAirport(dataFields[12]...
...);
                        }else if(a.airportExists(elementIds.get(0))){
                            originAirport = a.getAirport(elementIds.get...
...(0));
                        }else{
                            flightPlanErrors++;
                            continue; //return;
                        }

                        // Remove the origin airport from the route str...
...ing
                        if(a.airportExists(elementIds.get(0))){
                            elementIds.remove(0);
                            locations.remove(0);
                            elementClass.remove(0);
                        }

                        // Destination airport
                        String destinationAirportId = elementIds.get(el...
...ementIds.size()-1);
                        if(a.airportExists(destinationAirportId)){
                            destinationAirport = a.getAirport(destinati...
...onAirportId);

                            // Remove the destination airport from the ...
...route string
                            elementIds.remove(elementIds.size() - 1);
                            locations.remove(locations.size() - 1);
                            elementClass.remove(elementClass.size() - 1...
...);
                        }else{
                            flightPlanErrors++;
                            continue; //return;
                        }
```

```
                        // Use the preferred route in case of errors
                        String originAirportId = originAirport.getIdent...
...ifier();
                        if((!completeRouteInfo || elementIds.size() == ...
...0)
                        && a.hasPreferredRoute(originAirportId, destina...
...tionAirportId)){
                            route = a.getPreferredRoute(originAirportId...
..., destinationAirportId);

                            elementIds = new ArrayList<String>();
                            locations = new ArrayList<GPoint2D<Double>>...
...();
                            elementClass = new ArrayList<String>();
                            completeRouteInfo = FAPio.parseEtmsRoute(ro...
...ute,
                                    elementIds,locations,elementClass,f...
...alse, altitude);

                            // Remove origin and destination airport fr...
...om preferred route
                            elementIds.remove(0);
                            locations.remove(0);
                            elementClass.remove(0);
                            elementIds.remove(elementIds.size() - 1);
                            locations.remove(locations.size() - 1);
                            elementClass.remove(elementClass.size() - 1...
...);
                        }

                        if(completeRouteInfo){
                            Flight f = new Flight(aircraftId, originAir...
...port, destinationAirport,
                                    aircraftType, altitude);

                            // Get the departure time
                            GregorianCalendar departureTime = FAPio.get...
...Time(dataFields[14], dataFields[15]);
                            f.setDepartureTime(departureTime);

                            if(elementIds.size() > 0){
                                // Set the planned trajectory for the f...
...light
                                FTrajectory tPlanned = new FTrajectory(...
...);
                                f.setPlannedTrajectory(tPlanned);
```

```
                                // Add waypoint info
                                tPlanned.build4DTrajectoryFromWaypoints...
...(elementIds, locations, aircraftType);

                                // Add the flight
                                a.addFlight(f);
                            }else{
                                blankRoutes++;
                            }


                            if(originAirport == null || destinationAirp...
...ort == null){
                                System.out.println(line);
                                System.out.println(originAirport.getIde...
...ntifier());
                                System.out.println(elementIds);
                                System.out.println(destinationAirport.g...
...etIdentifier());
                            }
                        }else{
                            flightPlanErrors++;
                        }
                        i++;

                        elementIds = null;
                        locations = null;
                        elementClass = null;
                    }
                } else{
                    if(unknownType.containsKey(acType)){
                        int numTimes = unknownType.get(acType);
                        unknownType.put(acType, numTimes+1);
                        //unknownType.get(acType)++;
                    } else{
                        unknownType.put(acType, 1);
                    }
                    //System.out.println(line);
                    j++;
                }

                dataFields = null;
            }
        }

    }
```

```
    /** Read flight plan information into flight data structure.
     *  Currently only adds aircraft type information.
     */
    public static void readFlightPlanDataFromFlightExplorer(String file...
...Name, boolean useUniqueId) throws IOException{
        System.out.println(fileName);
        Airspace a = APData.airspace;

        readFlightPlanData.unknownType = new HashMap<String, Integer>(
                Settings.hashMapDefaultInitialCapacity,
                Settings.hashMapLoadFactor);


        BufferedReader d = new BufferedReader(new FileReader(fileName))...
...;
        String line;

        int lineCount = 0;
        int maxLines = Settings.flightPlanFileLinesToRead;
        System.out.println("Reading first " + Integer.toString(maxLines...
...) +
                " lines of the flight plan file.");

        System.out.println("NOT discarding flights less than 18,000 ft....
...");

        // ArrayList to keep track of thread
        ArrayList<Thread> threads = new ArrayList<Thread>();

        // Send 1000 lines to each thread to reduce the overhead incurr...
...ed by
        // creating too many threads.
        boolean firstRead = true;
        int lineIdx = -1;
        int linesToRead = 1000;
        ArrayList<String> lines = new ArrayList<String>(linesToRead);

        while((line = d.readLine()) != null){
            if(lineCount++ > maxLines) break;

            lineIdx++;

            if(firstRead && lineIdx < linesToRead-1){
                // Add up to next to last element
                lines.add(line);
```

```
            }else if(lineIdx < linesToRead-1){
                // Reuse the lines ArrayList after the first iteration
                lines.set(lineIdx, line);
            }else{
                if(firstRead){
                    lines.add(line);
                    firstRead = false;
                }else{
                    lines.set(lineIdx, line);
                }

                // Create a copy of the lines ArrayList
                ArrayList<String> lines2 = new ArrayList<String>(lines)...
...;

                Thread t = new Thread(new readFlightPlanData(lines2, us...
...eUniqueId));
                t.start();
                threads.add(t);

                lineIdx = -1;
            }
        }

        // Send remaining lines to a new thread
        while(lines.size() > lineIdx+1){
            lines.remove(lineIdx+1);
        }
        Thread t2 = new Thread(new readFlightPlanData(lines, useUniqueI...
...d));
        t2.start();
        threads.add(t2);

        try{
            for(Thread t:threads){
                t.join();
                //t = null;
            }
        }catch(InterruptedException e){
            System.out.println("Error reading flight plan data!");
        }


        d.close();

        line = null;
        readFlightPlanData.unknownType = null;
```

```
        System.out.println("Number of flight plans          : " +
                Integer.toString(readFlightPlanData.totalFlightPlans));...
...        System.out.println("Number of flight plan errors    : " +
                Integer.toString(readFlightPlanData.flightPlanErrors));...
...        System.out.println("Number of aircraft types found  : " + readF...
...lightPlanData.i);
        System.out.println("Number of unknown aircraft types: " + readF...
...lightPlanData.j);
        System.out.println("Number of flights without routes: " +
                Integer.toString(readFlightPlanData.blankRoutes));


    }

    /** Parses the ETMS altitude data
     *
     */
    public static int parseAltitude(String altitude){

        /* Do not include flights that:
                M = military
                VFR = VFR
                OTP = VFR conditions on top
                C = climbing
                ABV = above (not applicable in this case of IFR)
         */
        if(altitude.startsWith(APData.M) || altitude.startsWith(APData....
...VFR) || altitude.startsWith(APData.OTP)
        || altitude.startsWith(APData.ABV)){
            return 0;
        } else if(altitude.endsWith(APData.C)){
            altitude = altitude.split(APData.C)[0];
        } else if(altitude.endsWith(APData.T)){
            altitude = altitude.split(APData.T)[0];
        } else if(altitude.contains(APData.B)){
            altitude = altitude.split(APData.B)[0];
        }

        return Integer.parseInt(altitude);
    }

    public static GregorianCalendar getTime(String dateS, String timeS)...
...{
        SimpleDateFormat formatter = new SimpleDateFormat("MM/dd/yyyyHH...
...mmssz");
        String dateTime = dateS.concat(timeS+APData.UTC);     //08/29/2...
```

```
...005,000001

        ParsePosition pos = new ParsePosition(0);
        Date dt = formatter.parse(dateTime, pos);

        GregorianCalendar trackTime = new GregorianCalendar(TimeZone.ge...
...tTimeZone(APData.UTC));
        trackTime.setTime(dt);

        return trackTime;
    }

    /** Read radar track information into flight data structure.
     */
    public static void readRadarTrackDataFromFlightExplorer(String file...
...Name, boolean useUniqueId) throws IOException{
        System.out.println(fileName);
        Airspace a = APData.airspace;

        // UNKNOWN aircraft type
        EAircraft unknownAc = EAircraft.getAircraftType(APData.UNKNOWN)...
...;

        String line;

        int trackLineCount = 0;

        ArrayList<String> candidateFlights = new ArrayList<String>(1000...
...0);
        if(Settings.filterAltitude || Settings.filterTime){
            BufferedReader d = new BufferedReader(new FileReader(fileNa...
...me));
            while((line = d.readLine()) != null){
                trackLineCount++;

                if(trackLineCount > Settings.radarTrackFileLinesToRead)...
... break;

                String[] dataFields = line.split(APData.COMMA);
                String aircraftIdS = useUniqueId ? dataFields[6] + APDa...
...ta.UNDERSCORE + dataFields[0] :
                    dataFields[6];


//                if(!aircraftIdS.startsWith("JBU477") && !aircraftIdS....
...startsWith("JBU477")){
//                    continue;
```

```
//                } else{
//                    System.out.println(line);
//                }

                if(dataFields.length < 12) continue;
                int altitude = FAPio.parseAltitude(dataFields[9]);
                int hour = Integer.parseInt(dataFields[4].substring(0,2...
...));

                boolean inFilter = false;

                inFilter = Settings.filterAltitude ?
                    !(altitude < Settings.filterMinAltitude || altitude...
... > Settings.filterMaxAltitude)
                    : true;

                inFilter = Settings.filterTime ?
                    !(hour < Settings.filterMinTime || hour > Settings....
...filterMaxTime)
                    : inFilter;

                if(!inFilter){
                    continue;
                } else if(!candidateFlights.contains(aircraftIdS)){
                    candidateFlights.add(aircraftIdS);
                }
            }
            //System.out.print("Number of flights found: ");
            //System.out.println(candidateFlights.size());
            d.close();
        }

        BufferedReader d = new BufferedReader(new FileReader(fileName))...
...;

        int i = 0;

        int currentTrackLine = 0;

        boolean passedOnce = false;

        while((line = d.readLine()) != null){
            currentTrackLine++;

            if(currentTrackLine > Settings.radarTrackFileLinesToRead) b...
...reak;
```

```
            String[] dataFields = line.split(APData.COMMA);

            String aircraftIdS = useUniqueId ? dataFields[6] + APData.U...
...NDERSCORE + dataFields[0] :
                dataFields[6];
            String callSign = dataFields[6];
            /*if(aircraftIdS.startsWith("EGF515") == false){
                continue;
            }else{
                System.out.println(line);
            }*/

            /*if(aircraftIdS.startsWith("TRS494") || aircraftIdS.starts...
...With("JBU380")){
                System.out.println(line);
            }*/

            /*if(!aircraftIdS.startsWith("AAL884") && !aircraftIdS.star...
...tsWith("USA1057")){
                continue;
            }*/

            if(Settings.filterAltitude || Settings.filterTime){
                if(!candidateFlights.contains(aircraftIdS)){
                    continue;
                }
            }

            if(dataFields.length < 12) continue;    // Disregard header...
... lines
            GregorianCalendar trackTime = FAPio.getTime(dataFields[3], ...
...dataFields[4]);

            String groundSpeed = dataFields[8]; // Groundspeed empty or...
... 999 sometimes

            boolean isClimbing = dataFields[9].endsWith(APData.C);   //...
... Flight is climbing if altitude field ends with 'C'
            double altitudeD = (double)FAPio.parseAltitude(dataFields[9...
...]);
            if(altitudeD == 0.0){ continue; }   // VFR, OTP, etc. fligh...
...ts
            //if(altitudeD < 180){ continue; }


            double latitudeD = 0.0;
            String latitude = dataFields[10];
```

```
            String latitudeDeg = latitude.substring(0,2);
            String latitudeMin = latitude.substring(2,4);
            if(latitude.endsWith(APData.N)){
                //latitudeD = Double.parseDouble(latitude.split("N")[0]...
...)/60.0;
                latitudeD = Double.parseDouble(latitudeDeg) + (0.016666...
...7)*Double.parseDouble(latitudeMin);
            } else{
                System.out.println(line);
                continue;
            }

            double longitudeD = 0;
            String longitude = dataFields[11];
            String longitudeDeg = longitude.substring(0,3);
            String longitudeMin = longitude.substring(3,5);
            if(longitude.endsWith(APData.W)){
                //longitudeD = -1*Double.parseDouble(longitude.split("W...
...")[0])/60.0;
                longitudeD = -1*Double.parseDouble(longitudeDeg) - (0.0...
...166667)*Double.parseDouble(longitudeMin);
                //} else if(longitude.endsWith("E")){
                //    longitudeD = Double.parseDouble(longitude.split("...
...E")[0])/60.0;
            } else{
                //System.out.println(line);
                continue;
            }

            GPoint4D<Double> pt = new GPoint4D<Double>(longitudeD, lati...
...tudeD, altitudeD, trackTime);

            /* Check filter for track location - altitude considered ab...
...ove and below.
             */
            if(!a.flightInRange(longitudeD, latitudeD)){
                continue;
            }

            /* Create a new flight if not already defined */
            FTrajectory track;
            Flight f;
            if(a.containsFlight(aircraftIdS)){
                f = a.getFlight(aircraftIdS);
                track = f.getRadarTrackTrajectory();
            } else{
                if(a.containsFlight(callSign)){
```

```
                    f = a.getFlight(callSign);
                    f = new Flight(aircraftIdS, f.getAircraftType());
                }else{
                    a.addFlight(new Flight(callSign, unknownAc));
                    f = new Flight(aircraftIdS,unknownAc);
                }
                a.addFlight(f);
                track = new FTrajectory(trackTime);

                // Create a new trajectory object
                // using the first point as departure
                // time
                f.setRadarTrackTrajectory(track);
            }

            /* Check that aircraft has been set. */
            if(f.getAircraftType().getCallSign().equals(APData.UNKNOWN)...
... &&
                    a.numberComputerIdentifiersForCallsign(callSign) ==...
... 1){
                //String flightPlanIdentifier = callSign + a.firstCid(c...
...allSign);
                Flight fPlan = a.getFlight(callSign);
                f.addAircraftType(fPlan.getAircraftType());
            }

            try{
                track.addWaypoint(pt, f.getAircraftType());
            }catch(NullPointerException e){
                System.out.print("Flight: " + f.getCallSign());
                System.out.print("; Altitude: " + altitudeD);
                System.out.print("; Aircraft: ");
                System.out.println(f.getAircraftType().getCallSign());
            }


            track.addClimbingStatus(isClimbing);
            track.addGroundSpeed(groundSpeed);

            i++;
            if(i % 10000 == 0){
                System.out.print(i); System.out.print(" (");
                double progress = 100.0D*currentTrackLine/trackLineCoun...
...t;
                System.out.print(progress); System.out.println(" %)");

                //break;
```

```
            }
        }

        /*Table Name: Flight Track Information (TZ)
        Data Fields:
        FID  Unique ID
        Rule_No  Internal rule no.
        Message Type  TZ
        Message Time  GMT time message was sent
        Center  Identity of the FAA facility generating this message
        Aircraft ID  Aircraft identification
        CID -
        Ground Speed  Current Speed of aircraft relative to the surfac...
...e of the earth in knots
        Altitude  Altitude or flight level in hundreds of feet
        Latitude  Latitude of aircraft at Message Time
        Longitude  Longitude of aircraft at Message Time
         */

        d.close();

        // Persistence check
        System.out.println("Performing persistence check for NN0 flight...
... levels.");
        Iterator<String> e = a.getFlightIterator();
        while(e.hasNext()){
            Flight f = a.getFlight(e.next());
            FTrajectory t = f.getRadarTrackTrajectory();
            t.checkFlightLevelPersistence();
            if(t.numberOfWaypointsInAltitudeRange(Settings.filterMinAlt...
...itude, Settings.filterMaxAltitude) == 0){
                // Iterator does not allow changes to collection. Must ...
...use remove method
                //a.removeFlight(f.getCallSign());
                e.remove();
            }
        }

        /* Count number of flights with matched and unmatched aircraft ...
...*/
        Iterator<String> e2 = a.getFlightIterator();
        int numMatchedAircraft = 0;
        int numUnmatchedAircraft = 0;
        while(e2.hasNext()){
            Flight f = a.getFlight(e2.next());
            if(f.getAircraftType().getCallSign().equals(APData.UNKNOWN)...
...){
```

```
                numUnmatchedAircraft++;
            }else{ numMatchedAircraft++; }



        }
        System.out.println("Flights with matched aircraft: " + numMatch...
...edAircraft);
        System.out.println("Flights with unmatched aircraft: " + numUnm...
...atchedAircraft);
    }

    /* Writes the flight trajectory information for the purpose of inte...
...rpolating at 1 minute intervals.
     *
     * Coordinate Data:     Index, Latitude, Longitude, Time
     * Flight Index Data:   Flight Index, Start Index, End Index, Time ...
...Start, Time End
     *
     * Note: Time is not recorded at regular intervals - both index and...
... time info required
     */
    public static void writeRadarTracksForTimeInterpolation() throws IO...
...Exception{
        PrintWriter outCoordinates = new PrintWriter(new BufferedWriter...
...(new FileWriter(Settings.outRadarCoordinates)));
        PrintWriter outFlightIndex = new PrintWriter(new BufferedWriter...
...(new FileWriter(Settings.outFlightIndex)));
        PrintWriter outFlightCallsign = new PrintWriter(new BufferedWri...
...ter(new FileWriter(Settings.outFlightIndexCallsign)));
        PrintWriter outAltitudeStats = new PrintWriter(new BufferedWrit...
...er(new FileWriter(Settings.outAltitudeStats)));

        StringBuilder sStats = new StringBuilder();

        Airspace a = APData.airspace;
        Iterator<String> e = a.getFlightIterator();

        //int noFlights = a.noFlights();

        int idx = 1;
        int fl_idx = 1;
        while(e.hasNext()){
            /** Stringbuilder to hold file contents */
            StringBuilder sCoord = new StringBuilder();
            StringBuilder sIndex = new StringBuilder();
            StringBuilder sCallSign = new StringBuilder();
```

```
            Flight f = a.getFlight(e.next());
            FTrajectory t = f.getRadarTrackTrajectory();

            sIndex.append(fl_idx); sIndex.append(", ");
            sIndex.append(idx); sIndex.append(", ");

            sCallSign.append(fl_idx); sCallSign.append(", ");
            sCallSign.append(f.getCallSign()); sCallSign.append(APData....
...SLASH_N);

            // These two flights are very close.
            /*if(fl_idx == 167){
                System.out.print("Flight 1: "); // 59 = DAL949; 167 = T...
...RS494
                System.out.println(f.getCallSign());
            } else if(fl_idx == 177){
                System.out.print("Flight 2: "); // 168 = DAL1253; 177 =...
... JBU380
                System.out.println(f.getCallSign());
            }*/


            double min_t_minutes = 9999999999.9;
            double max_t_minutes = 0.0;
            double previousAltitude = 0.0;
            double currentAltitude = 0.0;
            double maxAltitude = 0.0;
            double previousTime = 0.0;
            boolean isClimbing = true;
            boolean isCruising = false;
            boolean isDescending = false;
            for (int i=0; i<t.noWaypoints(); i++){
                GPoint4D<Double> p = t.getWaypoint(i);

                if(p.getElevation() < 0.0D){
                    currentAltitude = previousAltitude;
                } else{
                    currentAltitude = p.getElevation();

                    maxAltitude = currentAltitude > maxAltitude ?
                        currentAltitude : maxAltitude;

                    isClimbing = (previousAltitude < currentAltitude);
                    isCruising = (previousAltitude == currentAltitude);...
...                    isDescending = (previousAltitude > currentAltitude)...
...;
```

```
                    previousAltitude = currentAltitude;

                }
                if(Settings.filterAltitude){
                    if(currentAltitude < Settings.filterMinAltitude || ...
...currentAltitude > Settings.filterMaxAltitude){
                        continue;
                    }
                }

                GregorianCalendar gc = p.getTime();
                double t_minutes = gc.get(gc.HOUR_OF_DAY)*60.0 + gc.get...
...(gc.MINUTE)*1.0 + gc.get(gc.SECOND)/60.0;

                if(Settings.filterTime){
                    if(t_minutes < Settings.filterMinTime || t_minutes ...
...> Settings.filterMaxTime){
                        continue;
                    }
                }

                // Write as different flights if points separated by mo...
...re than 10 minutes
                if(t_minutes - previousTime > 10.0D && previousTime > 0...
....0D){
                    sIndex.append(idx-1); sIndex.append(", ");
                    int min_t = t.noWaypoints() < 2 ? (int)Math.round(m...
...in_t_minutes) : (int)Math.ceil(min_t_minutes);
                    sIndex.append(min_t); sIndex.append(", ");
                    int max_t = Math.max(min_t, (int)Math.floor(max_t_m...
...inutes));
                    sIndex.append(max_t); sIndex.append(APData.SLASH_N)...
...;
                    fl_idx++;

                    /*if(fl_idx == 167){
                        System.out.print("Flight 1: "); // 59 = DAL949;...
... 167 = TRS494
                        System.out.println(f.getCallSign());
                    } else if(fl_idx == 177){
                        System.out.print("Flight 2: "); // 168 = DAL125...
...3; 177 = JBU380
                        System.out.println(f.getCallSign());
                    }*/

                    sIndex.append(fl_idx); sIndex.append(", ");
```

```
                    sIndex.append(idx); sIndex.append(", ");

                    sCallSign.append(fl_idx); sCallSign.append(", ");
                    sCallSign.append(f.getCallSign()); sCallSign.append...
...(APData.SLASH_N);

                    min_t_minutes = 9999999999.9;
                    max_t_minutes = 0.0;
                }

                min_t_minutes = t_minutes < min_t_minutes ? t_minutes :...
... min_t_minutes;
                max_t_minutes = t_minutes > max_t_minutes ? t_minutes :...
... max_t_minutes;

                sCoord.append(idx); sCoord.append(", ");
                sCoord.append(p.getLatitude()); sCoord.append(", ");
                sCoord.append(p.getLongitude()); sCoord.append(", ");
                sCoord.append(t_minutes); sCoord.append(", ");
                sCoord.append(t.getGroundSpeed(i)); sCoord.append(APDat...
...a.COMMA);

                // Leave a blank field for backwards compatibility
                sCoord.append("-999,");

                // Write aircraft alias number
                EAircraft acType = f.getAircraftType();
                sCoord.append(acType.getAliasNumber()); sCoord.append(A...
...PData.COMMA);

                // Write desired airspeed
                if(acType.getCallSign().length() == 0){
                    sCoord.append("-9999");
                } else if(isClimbing){
                    sCoord.append(acType.getClimbAirspeed(currentAltitu...
...de));
                } else if(isDescending){
                    sCoord.append(acType.getDescentAirspeed(currentAlti...
...tude));
                } else{   //if(isCruising) is default
                    sCoord.append(acType.getCruiseAirspeed(currentAltit...
...ude));
                }
                sCoord.append(APData.COMMA);

                // Include flight index for verification purposes
                sCoord.append(fl_idx); sCoord.append(APData.COMMA);
```

```
                // Add track altitude information
                sCoord.append(currentAltitude); sCoord.append(APData.SL...
...ASH_N);

                previousTime = t_minutes;
                idx++;
            }
            sIndex.append(idx-1); sIndex.append(", ");

            //long min_t = Math.round(min_t_minutes);
            //min_t = min_t < min_t_minutes ? min_t + 1 : min_t;
            int min_t = t.noWaypoints() < 2 ? (int)Math.round(min_t_min...
...utes) : (int)Math.ceil(min_t_minutes);
            sIndex.append(min_t); sIndex.append(", ");

            int max_t = Math.max(min_t, (int)Math.floor(max_t_minutes))...
...;
            sIndex.append(max_t); sIndex.append(APData.SLASH_N);


            fl_idx++;

            // Add statistics
            if(f.getCruisingFlightLevel() > 0){
                a.addFlightLevelStatistic(f.getCruisingFlightLevel(), m...
...axAltitude);
            }

            outCoordinates.print(sCoord.toString());
            outFlightIndex.print(sIndex.toString());
            outFlightCallsign.print(sCallSign.toString());
        }

        for(int i = 180; i < 410; i += 10){
            try{
                sStats.append(a.getFlightLevelStatistics(i));
            }catch(NullPointerException e2){
                int j = i;
            }
        }

        outCoordinates.close();
        outFlightIndex.close();
        outFlightCallsign.close();

        outAltitudeStats.print(sStats.toString());
```

```
        outAltitudeStats.close();
    }


    /*cos_AOB = cosd(lat1)*cosd(lat2)*cosd(lon2-lon1)+sind(lat1)*sind(l...
...at2)
     *AOB = acos(cos_AOB)
     *AOB*3440
     */
    public static void sectorDemandStatistics() throws IOException{
        final int INIT_NUM_FLIGHTS = 10000;

        //String fileName = "C:\\Documents and Settings\\Jeff\\My Docum...
...ents\\Drop Box\\Analysis\\Mesoscopic Flow Model\\Computational Results...
...\\Sector Entry Distribution\\00H To 24H FL350 UTC 050829\\flightevent_...
...10H_To_12H.out.1";
        //String fileName = "/Users/antoniotrani/Documents/Jeff/Mesosco...
...pic Flow Model/Computational Results/Sector Entry Distribution/00H To ...
...24H FL350 UTC 050829/flightevent.out.1";
        //BufferedReader d = new BufferedReader(new FileReader(fileName...
...));

        String line = APData.NULLSTRING;

        HashMap<String, ACenter> centers = new HashMap<String, ACenter>...
...(10,
                Settings.hashMapLoadFactor);

        HashMap<String, ArrayList<Integer>> flights = new HashMap<Strin...
...g, ArrayList<Integer>>(INIT_NUM_FLIGHTS,
                Settings.hashMapLoadFactor);
        HashMap<Integer, GPoint2D<Double>> locations = new HashMap<Inte...
...ger, GPoint2D<Double>>(INIT_NUM_FLIGHTS,
                Settings.hashMapLoadFactor);
        HashMap<Integer, String> times = new HashMap<Integer, String>(I...
...NIT_NUM_FLIGHTS,
                Settings.hashMapLoadFactor);
        HashMap<Integer, Integer> startHours = new HashMap<Integer, Int...
...eger>(INIT_NUM_FLIGHTS,
                Settings.hashMapLoadFactor);
        HashMap<Integer, ACenter> enterCenters = new HashMap<Integer, A...
...Center>(INIT_NUM_FLIGHTS,
                Settings.hashMapLoadFactor);
        HashMap<Integer, ASector> enterSectors = new HashMap<Integer, A...
...Sector>(INIT_NUM_FLIGHTS,
                Settings.hashMapLoadFactor);
```

```
        int k=0;

        String flightEventDirectory = "C:\\Documents and Settings\\Jeff...
...\\My Documents\\Drop Box\\Analysis\\Mesoscopic Flow Model\\Computation...
...al Results\\Sector Entry Distribution\\00H To 24H FL350 UTC 050829\\";...
...
        // Iterate through all ".PTF" file in BADA directory.
        File fileN = new File(flightEventDirectory);
        String[] fileList = fileN.list();
        for(int i=0; i<fileList.length; i++){
            if(!fileList[i].endsWith(".out.1") || !fileList[i].startsWi...
...th("flightevent_")){
                continue;
            }

            String fileName = flightEventDirectory + fileList[i];
            //System.out.println(fileName);
            BufferedReader d = new BufferedReader(new FileReader(fileNa...
...me));

            while((line = d.readLine()) != null){
                // Discard the header line
                if(line.startsWith("{1Time")) continue;

                String[] dataFields = line.split(";");

                // Only consider sector pierce events
                String event = dataFields[3];
                if(!(event.equals("#SP") || event.equals("#SX"))) conti...
...nue;

                // Only consider flights at 35,000 ft
                double flightLevel = Double.parseDouble(dataFields[18])...
...;
                if(flightLevel < 345.0 || flightLevel > 355.0) continue...
...;

                double latitude = Double.parseDouble(dataFields[16]);
                double longitude = Double.parseDouble(dataFields[17]);
                GPoint2D<Double> location = new GPoint2D<Double>(longit...
...ude, latitude);

                String time = dataFields[0];
                int startHour = Integer.parseInt(time.split(APData.COLO...
...N)[0]);

                String callSign = dataFields[1];
```

```
                String enterCenter = dataFields[4];
                String enterSector = dataFields[5];

                if(enterCenter.startsWith("Null") || enterSector.starts...
...With("Null")) continue;

                //if(!enterCenter.startsWith("ZNY")) continue;

                ACenter center;
                if(centers.containsKey(enterCenter)){
                    center = centers.get(enterCenter);
                } else{
                    center = new ACenter(enterCenter);
                    centers.put(enterCenter, center);
                }

                center.addHourlyDemand(startHour);

                ASector sector;
                if(center.containsSector(enterSector)){
                    sector = center.getSector(enterSector);
                } else{
                    sector = new ASector(enterSector);
                    center.addSector(sector);
                }

                sector.addDemandAtPoint(location, startHour);

                // Add data to lists
                if(flights.containsKey(callSign)){
                    ArrayList<Integer> idxs = flights.get(callSign);
                    idxs.add(k);
                } else{
                    ArrayList<Integer> idxs = new ArrayList<Integer>();...
...                     idxs.add(k);
                    flights.put(callSign, idxs);
                }
                locations.put(k, location);
                times.put(k, time);
                startHours.put(k, startHour);
                enterCenters.put(k, center);
                enterSectors.put(k, sector);
                k++;
            }

            d.close();
        }
```

```
        // Write demand data
        PrintWriter outDemand = new PrintWriter(new BufferedWriter(new ...
...FileWriter(flightEventDirectory + "sector_jet_route_demand_results.dat...
..."")));
        StringBuilder sDemand = new StringBuilder();

        sDemand.append("Time,Flight,Hour,Center,CenterHourlyDemand,Sect...
...or,SectorEntryDemand,SectorExitDemand,SectorAverageDemand,SectorHourly...
...Demand\n");

        Iterator<String> e = flights.keySet().iterator();
        while(e.hasNext()){
            String callSign = e.next();
            ArrayList<Integer> idxs = flights.get(callSign);

            for(int j = 0; j < idxs.size(); j++){
                int idx = idxs.get(j);
                GPoint2D<Double> location = locations.get(idx);

                String time = times.get(idx);
                sDemand.append(times.get(idx) + APData.COMMA + callSign...
... + APData.COMMA);


                int startHour = startHours.get(idx);
                sDemand.append(startHour); sDemand.append(APData.COMMA)...
...;

                ACenter center = enterCenters.get(idx);
                sDemand.append(center.getName() + APData.COMMA);
                sDemand.append(center.getHourlyDemand(startHour)); sDem...
...and.append(APData.COMMA);

                ASector sector = enterSectors.get(idx);
                sDemand.append(sector.getName() + APData.COMMA);
                int sectorEntryDemand = sector.getHourlyDemandAtPoint(l...
...ocation, startHour);
                sDemand.append(sectorEntryDemand); sDemand.append(APDat...
...a.COMMA);

                if(j+1 < idxs.size() && sector.getName().equals(enterSe...
...ctors.get(idxs.get(j+1)).getName())){
                    // This is the demand for sector exit if pt is sect...
...or entry.
                    GPoint2D<Double> location2 = locations.get(idxs.get...
```

```
...(j+1));
                    int sectorExitDemand = sector.getHourlyDemandAtPoin...
...t(location2, startHour);
                    sDemand.append(sectorExitDemand); sDemand.append(AP...
...Data.COMMA);
                    double sectorAverageDemand = 0.5D*((double)sectorEn...
...tryDemand + (double)sectorExitDemand);
                    sDemand.append(sectorAverageDemand);
                    j++;
                }else if(j-1 >= 0 && sector.getName().equals(enterSecto...
...rs.get(idxs.get(j-1)).getName())){
                    // This is the demand for sector entry if pt is sec...
...tor exit.
                    GPoint2D<Double> location2 = locations.get(idxs.get...
...(j-1));
                    int sectorExitDemand = sector.getHourlyDemandAtPoin...
...t(location2, startHour);
                    sDemand.append(sectorExitDemand); sDemand.append(AP...
...Data.COMMA);
                    double sectorAverageDemand = 0.5D*((double)sectorEn...
...tryDemand + (double)sectorExitDemand);
                    sDemand.append(sectorAverageDemand);
                }else{
                    sDemand.append("-1,");
                    sDemand.append(sectorEntryDemand);
                }

                sDemand.append(APData.COMMA); sDemand.append(sector.get...
...HourlyDemand(startHour));

                sDemand.append(APData.SLASH_N);

            }
        }

        outDemand.print(sDemand.toString());
        outDemand.close();
    }

}
```

FAPioAircraftReader.java

```
/*
 * FAPioAircraftReader.java
 *
 * Created on May 27, 2007, 1:36 AM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author Jeff
 */
import java.util.*;
import java.io.*;
public class FAPioAircraftReader {

    /** Creates a new instance of FAPioAircraftReader */
    public FAPioAircraftReader() {
    }

    /** Read aircraft types data from RAMS input file */
    public FAPioAircraftReader readAircraftTypes(String fileName) throw...
...s IOException{
        //Airspace a = APData.airspace;

        BufferedReader d = new BufferedReader(new FileReader(fileName))...
...;

        d.readLine();     // Do not use header line. ie { ACPERFORMANCE...
... based on BADA 3.5 }

        String line;
        while((line = d.readLine()) != null){
            String[] dataFields = line.split(APData.SPACE);
            int acAlias = Integer.parseInt(dataFields[1]);
            String aircraftId = new String(dataFields[0]);
            EAircraft ac = new EAircraft(aircraftId, acAlias);

            EAircraft.addAircraftType(ac);

            dataFields = null;
        }
        line = null;

        d.close();
```

```
        // Add an unknown aircraft type
        EAircraft.addAircraftType(new EAircraft());

        return this;
    }

    /** Read preprocessed data */
    public FAPioAircraftReader readPreprocessedClimbAndDescent(String f...
...ileName) throws IOException{
        BufferedReader d = new BufferedReader(new FileReader(fileName))...
...;

        String line;
        while((line = d.readLine()) != null){
            String[] dataFields = line.split(APData.COMMA);

            EAircraft ac = EAircraft.getAircraftType(dataFields[0]);

            // Format HH:MM:SS
            String[] timeS = dataFields[1].split(APData.COLON);
            double time = Double.parseDouble(timeS[0])*60.0D + Double.p...
...arseDouble(timeS[1]) +
                    Double.parseDouble(timeS[2])*(1/60.0D);

            double altitude = Double.parseDouble(dataFields[8]);
            double distance = Double.parseDouble(dataFields[9]);

            if(dataFields[2].equals(APData.CLIMB)){
                ac.addPreprocessedClimb(altitude, time, distance);
            }else if(dataFields[2].equals(APData.DESCENT)){
                ac.addPreprocessedDescent(altitude, time, distance);
            }

            dataFields = null;
        }
        line = null;


        d.close();


        // Iterate through aircraft and set the array for Matlab
        Iterator<String> e = EAircraft.getAircraftTypeIterator();
        while(e.hasNext()){
            EAircraft ac = EAircraft.getAircraftType(e.next());
            ac.calculatePreprocessedClimbArray();
```

```
            ac.calculatePreprocessedDescentArray();
        }
        return this;
    }

    /** Read BADA data */
    public FAPioAircraftReader readBada() throws IOException{
        //Airspace a = APData.airspace;

        // Statistics for "UNKNOWN" aircraft type
        EAircraft unknownAc = EAircraft.getAircraftType(APData.UNKNOWN)...
...;
        double maxRateOfClimb = -1.0D;
        double maxRateOfDescent = -1.0D;

        // Iterate through all ".PTF" file in BADA directory.
        File fileN = new File(Settings.badaDirectory);
        String[] fileList = fileN.list();
        for(int i=0; i<fileList.length; i++){
            if(!fileList[i].endsWith(APData.PTF)){
                continue;
            }

            String fileName = Settings.badaDirectory + fileList[i];
            //System.out.println(fileName);
            BufferedReader d = new BufferedReader(new FileReader(fileNa...
...me));

            String line = APData.NULLSTRING;
            //String acType = "";
            EAircraft ac = null;

            while((line = d.readLine()) != null){
                if(line.length() < 3){
                    continue;
                }

                if(line.startsWith(APData.AC_TYPE)){
                    String acType = line.split(APData.COLON)[1];
                    acType = acType.split(APData.UNDERSCORE)[0];
                    acType = acType.trim();

                    if(acType.startsWith(APData.FGTH) || acType.startsW...
...ith(APData.FGTL)
                    || acType.startsWith(APData.FGTN) || acType.startsW...
...ith(APData.NIMBUSTR)
                    || acType.startsWith(APData.SR22G2) || acType.start...
```

```
...sWith(APData.VLJ)){
                        break;
                    }
                    ac = EAircraft.getAircraftType(acType);
                }

                if(line.substring(2,3).equalsIgnoreCase(APData.ZERO) ||...
...                         line.substring(2,3).equalsIgnoreCase(APData.FIV...
...E)){
                    if(line.length() < 80){
                        continue;
                    }
                    double flightLevel = Double.parseDouble(line.substr...
...ing(0,3).trim());

                    /** Rate of Climb/Descent info */
                    double rocLo = Double.parseDouble(line.substring(41...
...,45).trim());
                    double rocNom = Double.parseDouble(line.substring(4...
...6,50).trim());
                    double rocHi = Double.parseDouble(line.substring(51...
...,55).trim());
                    double rodNom = Double.parseDouble(line.substring(7...
...3,77).trim());

                    rocNom = Math.max(Math.max(rocLo, rocHi),rocNom);

                    /** Set statistics for "UNKNOWN" aircraft type */
                    if(rocNom > unknownAc.getMaxRateOfClimb(flightLevel...
...)){
                        unknownAc.addRateOfClimb(flightLevel, rocNom);
                    }
                    if(rodNom > unknownAc.getMaxRateOfDescent(flightLev...
...el)){
                        unknownAc.addRateOfDescent(flightLevel, rodNom)...
...;
                    }

                    /** Statistics for max roc/rod */
                    maxRateOfClimb = rocNom > maxRateOfClimb ? rocNom :...
... maxRateOfClimb;
                    maxRateOfDescent = rodNom > maxRateOfDescent ? rodN...
...om : maxRateOfDescent;

                    /** TAS [kts] */
                    double climbAirspeed = Double.parseDouble(line.subs...
...tring(35,38).trim());
```

```
                    double descentAirspeed = Double.parseDouble(line.su...
...bstring(68,71).trim());
                    double cruiseAirspeed = flightLevel < 50.0D ? climb...
...Airspeed :
                        Double.parseDouble(line.substring(7,10).trim())...
...;

                    try{
                        ac.addRateOfClimb(flightLevel, rocNom);
                        ac.addRateOfDescent(flightLevel, rodNom);
                        ac.addCruiseAirspeed(flightLevel, cruiseAirspee...
...d);
                        ac.addClimbAirspeed(flightLevel, climbAirspeed)...
...;
                        ac.addDescentAirspeed(flightLevel, descentAirsp...
...eed);

                        // Add info to other similar aircraft
                        ArrayList<String> aircraftAliasList = EAircraft...
....aircraftAliasList(ac.getAliasNumber());

                        for(int j=0; j<aircraftAliasList.size(); j++){
                            EAircraft ac2 = EAircraft.getAircraftType(a...
...ircraftAliasList.get(j));

                            ac2.addRateOfClimb(flightLevel, rocNom);
                            ac2.addRateOfDescent(flightLevel, rodNom);
                            ac2.addCruiseAirspeed(flightLevel, cruiseAi...
...rspeed);
                            ac2.addClimbAirspeed(flightLevel, climbAirs...
...peed);
                            ac2.addDescentAirspeed(flightLevel, descent...
...Airspeed);
                        }

                    } catch(NullPointerException e){
                        System.out.println("Error in file: " + fileName...
...);
                    }
                }
            }
            line = null;
            d.close();
        }

        return this;
    }
```

```
    public FAPioAircraftReader readClimbDescentDistanceTime(String file...
...Name) throws IOException{
        BufferedReader d = new BufferedReader(new FileReader(fileName))...
...;

        String line;
        while((line = d.readLine()) != null){
            String[] dataFields = line.split(APData.COMMA);

            EAircraft ac = EAircraft.getAircraftType(dataFields[0]);

            double flightLevel = Double.parseDouble(dataFields[1]);

            double climbDistance = Double.parseDouble(dataFields[2]);
            double climbTime = Double.parseDouble(dataFields[3]);
            double descentDistance = Double.parseDouble(dataFields[4]);...
...             double descentTime = Double.parseDouble(dataFields[5]);

            ac.addClimbDistance(flightLevel, climbDistance);
            ac.addClimbTime(flightLevel, climbTime);
            ac.addDescentDistance(flightLevel, descentDistance);
            ac.addDescentTime(flightLevel, descentTime);

            dataFields = null;
        }
        line = null;


        d.close();

        return this;
    }
}
```

FAPioAircraftWriter.java

```
/*
 * FAPioAircraftWriter.java
 *
 * Created on May 28, 2007, 12:20 AM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author Jeff
 */
import java.util.*;
import java.io.*;
import java.text.*;
public class FAPioAircraftWriter {

    /** Creates a new instance of FAPioAircraftWriter */
    public FAPioAircraftWriter() {
    }

    /** Write a copy of the aircraft information for verification purpo...
...ses */
    public FAPioAircraftWriter writeAircraftTypesCopy(String fileName) ...
...throws IOException{
        PrintWriter out = new PrintWriter(new BufferedWriter(new FileWr...
...iter(fileName + ".cpy")));

        Iterator<String> i = EAircraft.getAircraftTypeIterator();
        while(i.hasNext()){
            out.println(i.next());
        }

        out.close();

        return this;
    }

    /** Write the results of the climb/descent time/distance */
    public FAPioAircraftWriter writeClimbDescentTimeDistance(String fil...
...eName) throws IOException{
        PrintWriter out = new PrintWriter(new BufferedWriter(new FileWr...
...iter(fileName)));

        DecimalFormat format = new DecimalFormat("###0.00000");
```

```
        StringBuffer s = new StringBuffer();

        Iterator<String> e = EAircraft.getAircraftTypeIterator();
        while(e.hasNext()){
            String aircraftS = e.next();
            EAircraft ac = EAircraft.getAircraftType(aircraftS);

            double flightLevel = 10.0D;
            while(ac.getClimbDistance(flightLevel) > 0.0D){
                s.append(aircraftS);
                s.append(",");
                s.append(format.format(flightLevel));
                s.append(",");
                s.append(format.format(ac.getClimbDistance(flightLevel)...
...));
                s.append(",");
                s.append(format.format(ac.getClimbTime(flightLevel)));
                s.append(",");
                s.append(format.format(ac.getDescentDistance(flightLeve...
...l)));
                s.append(",");
                s.append(format.format(ac.getDescentTime(flightLevel)))...
...;
                s.append("\n");

                flightLevel += 10.0D;
            }

        }

        out.print(s.toString());

        out.close();

        return this;
    }

}
```

FAPioAirspaceReader.java

```
/*
 * FAPioAirspaceReader.java
 *
 * Created on May 26, 2007, 2:13 AM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author Jeff
 */
import java.util.*;
import java.io.*;
public class FAPioAirspaceReader {
    private final Airspace a;

    /** Creates a new instance of FAPioAirspaceReader */
    public FAPioAirspaceReader(Airspace airspace) {
        a = airspace;
    }

    /** Read international waypoints and store them as fixes */
    public FAPioAirspaceReader readInternationalWaypoints(String fileNa...
...me) throws IOException{
        BufferedReader d = new BufferedReader(new FileReader(fileName))...
...;

        String line;
        while ((line=d.readLine()) != null){
            if(line.startsWith("WPT_IDENT")) continue;

            String[] dataFields = line.split(APData.COMMA);
            String id = dataFields[0];
            String country = dataFields[1];
            double latitude = Double.parseDouble(dataFields[2]);
            double longitude = Double.parseDouble(dataFields[3]);

            String waypointId = country.equals("US") ? id :
                id + APData.UNDERSCORE + country;

            if(!a.hasFix(waypointId) && !a.hasNavaid(waypointId)){
                a.addFix(new GPoint2D<Double>(longitude, latitude), way...
...pointId);
            }
```

```
        }

        line = null;
        d.close();


        return this;
    }

    /** Reads other airspace locations not covered in the other files *...
.../
    public FAPioAirspaceReader readNatfix(String fileName) throws IOExc...
...eption{
        BufferedReader d = new BufferedReader(new FileReader(fileName))...
...;

        String line;
        while ((line=d.readLine()) != null){
            String[] dataFields = line.split(APData.COMMA);

            if(dataFields.length < 4) continue;

            String identifier = new String(dataFields[0].trim());
            identifier = identifier;
            String type = new String(dataFields[3].trim());

            // Convert lat/lon
            Double lat = Double.parseDouble(dataFields[1]);
            Double lon = Double.parseDouble(dataFields[2]);

            if(type.equals(APData.ARPT) && !a.airportExists(identifier)...
...){
                a.addAirport(new Airport(identifier, new GPoint3D<Doubl...
...e>(lon, lat, 0D)));
            }else if(type.equals(APData.COORDN_) || type.equals(APData....
...GPS_WP) ||
                    type.equals(APData.MIL_REP) || type.equals(APData.M...
...IL_WAY) ||
                    type.equals(APData.NRS_WAY) || type.equals(APData.R...
...ADAR) ||
                    type.equals(APData.RNAV_WP) || type.equals(APData.W...
...AYPOIN)){
                if(!a.hasFix(identifier)){
                    a.addFix(new GPoint2D<Double>(lon, lat), identifier...
...);
                }
```

```
            }else if(!a.hasNavaid(identifier)){  // NAVAID
                a.addNavaid(new GPoint2D<Double>(lon, lat), identifier)...
...;
            }

            dataFields = null;
        }

        line = null;

        d.close();

        return this;
    }


    /** Reads and stores STARs */
    public FAPioAirspaceReader readStars(String fileName) throws IOExce...
...ption{
        BufferedReader d = new BufferedReader(new FileReader(fileName))...
...;

        String line;
        while ((line=d.readLine()) != null){
            String[] dataFields = line.split(APData.COMMA);

            if(dataFields.length < 4) continue;

            String identifier = dataFields[2];
            //int starLength = dataFields[3].trim().length();

            if(!a.hasFix(identifier) && !a.hasNavaid(identifier) &&
                    !a.airportExists(identifier)){
                //System.out.println(identifier); System.out.println(li...
...ne);
            }else{
                // Add STAR/DP
                String star = new String(dataFields[3]);
                Double lat = Double.parseDouble(dataFields[0]);
                Double lon = Double.parseDouble(dataFields[1]);
                a.addStar(star, identifier, new GPoint2D<Double>(lon,la...
...t));
            }

            dataFields = null;
```

```
        }

        line = null;

        d.close();

        return this;
    }

    /** Reads DAFIF international airways ignoring domestic airways. */...
...    public FAPioAirspaceReader readInternationalAirways(String fileName...
...) throws IOException{
        // Create list of domestic airways that will not be read from f...
...ile.
        ArrayList<String> domesticAwyIds = new ArrayList<String>();
        Iterator<String> e = a.getJetRouteIterator();
        while(e.hasNext()){
            domesticAwyIds.add(e.next());
        }
        e = a.getVictorAirwayIterator();
        while(e.hasNext()){
            domesticAwyIds.add(e.next());
        }
        e = a.getRnavRouteIterator();
        while(e.hasNext()){
            domesticAwyIds.add(e.next());
        }

        // Reader for the international airways file
        BufferedReader d = new BufferedReader(new FileReader(fileName))...
...;

        String line;

        // First get the number of points in the eastbound and westboun...
...d directions
        // so that the maximum waypoints in one direction may be import...
...ed
        HashMap<String, Integer> eastboundAirways = new HashMap<String,...
... Integer>();
        HashMap<String, Integer> westboundAirways = new HashMap<String,...
... Integer>();
        while((line=d.readLine()) != null){
            if(line.startsWith("ATS_IDENT")) continue;

            int firstComma = line.indexOf(APData.COMMA);
            int secondComma = line.indexOf(APData.COMMA, firstComma+1);...
```

```
...
            //String[] dataFields = line.split(APData.COMMA);
            //String airwayId = dataFields[0];
            String airwayId = line.substring(0, firstComma);

            HashMap<String, Integer> directionalAirways = null;
            String dir = line.substring(firstComma+1, secondComma);
            if(dir.equals("E")){  // Eastbound
                directionalAirways = eastboundAirways;
            }else if(dir.equals("W")){    // Westbound
                directionalAirways = westboundAirways;
            }

            int count = directionalAirways.containsKey(airwayId) ?
                directionalAirways.get(airwayId) : 0;

            count++;
            directionalAirways.put(airwayId, count);

            line = null;
        }

        d.close();

        d = new BufferedReader(new FileReader(fileName));
        int[] cL = new int[9];
        while((line=d.readLine()) != null){
            if(line.startsWith("ATS_IDENT")) continue;

            for(int i=0; i<cL.length; i++){
                if(i == 0){
                    cL[i] = line.indexOf(APData.COMMA);
                }else{
                    cL[i] = line.indexOf(APData.COMMA, cL[i-1]+1);
                }
            }

            //String[] dataFields = line.split(APData.COMMA);

            // Check that airway is not contained in the domestic airwa...
...ys list
            String airwayId = line.substring(0, cL[0]); //dataFields[0]...
...;
            if(domesticAwyIds.contains(airwayId)) continue;

            // Check for direction with the most points
            int eastboundAirwayPoints = eastboundAirways.containsKey(ai...
```

```
...rwayId) ?
                eastboundAirways.get(airwayId) : 0;
            int westboundAirwayPoints = westboundAirways.containsKey(ai...
...rwayId) ?
                westboundAirways.get(airwayId) : 0;

            // Only consider segments in the direction with the most po...
...ints
            String dir = line.substring(cL[0]+1, cL[1]);
            if(dir.equals("E") && westboundAirwayPoints > eastboundAirw...
...ayPoints
                    || dir.equals("W") && eastboundAirwayPoints >=
                    westboundAirwayPoints) continue;

            // Get information for the first waypoint
            String dataFields2 = line.substring(cL[1]+1, cL[2]);
            String dataFields3 = line.substring(cL[2]+1, cL[3]);
            String wayPointId1 = dataFields3.equals(APData.US) ? dataFi...
...elds2 :
                dataFields2 + APData.UNDERSCORE + dataFields3;
            wayPointId1 = wayPointId1;
            String wayPointId1_ = dataFields2;


            double lat1 = Double.parseDouble(line.substring(cL[3]+1,cL[...
...4])); //dataFields[4]);
            double lon1 = Double.parseDouble(line.substring(cL[4]+1,cL[...
...5])); //dataFields[5]);
            GPoint2D<Double> pt1 = new GPoint2D<Double>(lon1,lat1);

            // Add the first waypoint as a fix
            if(!a.hasFix(wayPointId1) && !a.hasNavaid(wayPointId1)){
                a.addFix(pt1, wayPointId1);
            }

            // Also add without country suffix if point does not exist
//            if(!a.hasFix(wayPointId1_) && !a.hasNavaid(wayPointId1_))...
...{
//                a.addFix(pt1, wayPointId1_);
//            }

            // Add the point to the airway
            a.addAirwayPoint(pt1, wayPointId1, airwayId);

            // Get information for the second waypoint
            String dataFields7 = line.substring(cL[6]+1,cL[7]);
            String dataFields6 = line.substring(cL[5]+1,cL[6]);
```

```
            String wayPointId2 = dataFields7.equals(APData.US) ? dataFi...
...elds6 :
                dataFields6 + APData.UNDERSCORE + dataFields7;
            String wayPointId2_ = dataFields6;


            double lat2 = Double.parseDouble(line.substring(cL[7]+1,cL[...
...8]));//dataFields[8]);
            double lon2 = Double.parseDouble(line.substring(cL[8]+1,lin...
...e.length()));//dataFields[9]);
            GPoint2D<Double> pt2 = new GPoint2D<Double>(lon2,lat2);

            // Add the second waypoint as a fix
            if(!a.hasFix(wayPointId2) && !a.hasNavaid(wayPointId2)){
                a.addFix(pt2, wayPointId2);
            }

            // Also add without country suffix if point does not exist
//            if(!a.hasFix(wayPointId2_) && !a.hasNavaid(wayPointId2_))...
...{
//                a.addFix(pt2, wayPointId2_);
//            }

            // Add the point to the airway
            a.addAirwayPoint(pt2, wayPointId2, airwayId);

            //dataFields = null;
            line = null;
        }

        d.close();

        domesticAwyIds = null;


        return this;
    }

    public enum AirwayDirectionComparison{
        DifferentDirection, SameDirection, NotImported
    }

    /** Reads airways from file into a data structure */
    public FAPioAirspaceReader readAirways(String fileName) throws IOEx...
...ception{
        // Reader for the international airways file
        BufferedReader d = new BufferedReader(new FileReader(fileName))...
```

```
...;

        String line;
        String previousPointDesignator = null;

        HashMap<String, AirwayDirectionComparison> isSameDirection = ne...
...w
                HashMap<String, AirwayDirectionComparison>();

        int previousIndex = 0;
        int currentIndex = 0;

        // Read once to get direction
        while((line=d.readLine()) != null){
            String[] dataFields = line.split(APData.COMMA);

            String airwayDesignator = dataFields[0].trim();

            /* Leading zeros in the designator must be removed.
               e.g. J006 should be J6
             */
            while(airwayDesignator.substring(1,2).equals(APData.ZERO)){...
...                 airwayDesignator = airwayDesignator.replaceFirst(APData...
....ZERO,
                        APData.NULLSTRING);
            }

            /* Airway type
             * A = Alaska Airway
             * H = Hawaii VOR Airway
             */
            String airwayType = dataFields[1].trim();
            airwayDesignator = airwayDesignator + airwayType;

            // Set flag of not imported initially
            if(!isSameDirection.containsKey(airwayDesignator)){
                isSameDirection.put(airwayDesignator,
                        AirwayDirectionComparison.NotImported);

                previousIndex = -1;
                currentIndex = -1;
            }

            String ptDesignator = dataFields[4];
            if(a.hasAirway(airwayDesignator)){
                Airway awy = a.getAirway(airwayDesignator);
```

```
                if(awy.getPointIndex(ptDesignator) > -1){
                    currentIndex = awy.getPointIndex(ptDesignator);

                    previousPointDesignator = ptDesignator;

                    if(previousIndex > -1 && isSameDirection.get(airway...
...Designator).equals(
                            AirwayDirectionComparison.NotImported)){
                        if(previousIndex < currentIndex){
                            isSameDirection.put(airwayDesignator,
                                    AirwayDirectionComparison.SameDirec...
...tion);
                        }else if(previousIndex > currentIndex){
                            isSameDirection.put(airwayDesignator,
                                    AirwayDirectionComparison.Different...
...Direction);
                        }
                    }

                    previousIndex = currentIndex;
                }
            }


        }

        d.close();
        d = new BufferedReader(new FileReader(fileName));

        // Read again to add the airway points
        while ((line=d.readLine()) != null){
            String[] dataFields = line.split(APData.COMMA);

            String airwayDesignator = dataFields[0].trim();

            /* Leading zeros in the designator must be removed.
               e.g. J006 should be J6
             */
            while(airwayDesignator.substring(1,2).equals(APData.ZERO)){...
...                 airwayDesignator = airwayDesignator.replaceFirst(APData...
....ZERO,
                        APData.NULLSTRING);
            }


            /* Airway type
             * A = Alaska Airway
```

```
             * H = Hawaii VOR Airway
             */
            String airwayType = dataFields[1].trim();
            airwayDesignator = airwayDesignator + airwayType;

            String ptType = dataFields[3];
            String ptDesignator = dataFields[4];
            GPoint2D<Double> pt;
            if(ptType == APData.NAVAID && a.hasNavaid(ptDesignator)){
                pt = a.getNavaid(ptDesignator);
            } else if(ptType == APData.FIX && a.hasFix(ptDesignator)){
                pt = a.getFix(ptDesignator);
            } else{ // Add point as a fix
                Double lat = Double.parseDouble(dataFields[5]);
                Double lon = Double.parseDouble(dataFields[6]);
                pt = new GPoint2D<Double>(lon, lat);
                a.addFix(pt, ptDesignator);
            }

            AirwayDirectionComparison adc = isSameDirection.get(airwayD...
...esignator);
            // Airway was not imported with international airways
            if(adc.equals(AirwayDirectionComparison.NotImported)){
                a.addAirwayPoint(pt, ptDesignator, airwayDesignator);

                if(airwayDesignator.startsWith(APData.Q) && airwayDesig...
...nator.endsWith(APData.R)){
                    a.addAirwayPoint(pt, ptDesignator, airwayDesignator...
....replace(APData.R,
                            APData.NULLSTRING));
                }
            }else{ // Airway was imported with international airways

                boolean addAfter = adc.equals(AirwayDirectionComparison...
....SameDirection);

                a.addAirwayPoint(pt, ptDesignator, airwayDesignator, ad...
...dAfter, previousPointDesignator);

                if(airwayDesignator.startsWith(APData.Q) && airwayDesig...
...nator.endsWith(APData.R)){
                    a.addAirwayPoint(pt, ptDesignator, airwayDesignator...
....replace(APData.R,
                            APData.NULLSTRING), addAfter, previousPoint...
...Designator);
                }
            }
```

```
            previousPointDesignator = ptDesignator;

            dataFields = null;
        }

        d.close();

        line = null;

        return this;
    }


    /** Reads navigational fix into a data structure */
    public FAPioAirspaceReader readFixes(String fileName) throws IOExce...
...ption{
        BufferedReader d = new BufferedReader(new FileReader(fileName))...
...;

        String line;
        while ((line=d.readLine()) != null){
            String[] dataFields = line.split(APData.COMMA);

            if(dataFields.length < 4) continue;

            String identifier = new String(dataFields[0]);
            identifier = identifier.trim(); // Remove leading and trail...
...ing whitespace

            Double lon = Double.parseDouble(dataFields[2]);
            Double lat = Double.parseDouble(dataFields[1]);
            GPoint2D<Double> pt = new GPoint2D<Double>(lon, lat);

            a.addFix(pt, identifier);

            String description = new String(dataFields[3]);
            a.addFixType(description, identifier);

            dataFields = null;
        }

        d.close();

        line = null;

        return this;
```

```
    }


    /** Reads NAVAID information into data structure */
    public FAPioAirspaceReader readNavaids(String fileName) throws IOEx...
...ception{
        BufferedReader d = new BufferedReader(new FileReader(fileName))...
...;

        String line;
        while ((line=d.readLine()) != null){
            String[] dataFields = line.split(APData.COMMA);

            String identifier = new String(dataFields[0]);
            identifier = identifier.trim(); // Remove leading and trail...
...ing whitespace

            Double lon = Double.parseDouble(dataFields[2]);
            Double lat = Double.parseDouble(dataFields[1]);
            GPoint2D<Double> pt = new GPoint2D<Double>(lon, lat);

            a.addNavaid(pt, identifier);

            dataFields = null;
        }

        d.close();

        line = null;

        return this;
    }

    /** Reads airport type information into a data structure */
    public FAPioAirspaceReader readAirportTypes(String fileName) throws...
... IOException{
        BufferedReader d = new BufferedReader(new FileReader(fileName))...
...;

        String line;
        while((line=d.readLine()) != null){
            String[] dataFields = line.split(APData.COMMA);

            String airport1 = dataFields[0].trim();
            String airport2 = dataFields[1].trim();
            String aptType = dataFields[2].trim();
```

```
            if(!a.airportExists(airport1) || !a.airportExists(airport2)...
...) continue;

            Airport apt1 = a.getAirport(airport1);
            Airport apt2 = a.getAirport(airport2);
            if(aptType.equals("Large") && airport1.length() == 3){
                apt1.setAirportType(airport1);
                apt2.setAirportType(airport1);
            }else if(aptType.equals("Large") && airport2.length() == 3)...
...{
                apt1.setAirportType(airport2);
                apt2.setAirportType(airport2);
            }else{
                apt1.setAirportType(aptType);
                apt2.setAirportType(aptType);
            }
        }

        line = null;

        d.close();

        return this;

    }

    /** Reads airport information into data structure */
    public FAPioAirspaceReader readAirports(String fileName) throws IOE...
...xception{
        BufferedReader d = new BufferedReader(new FileReader(fileName))...
...;

        String line;
        while ((line=d.readLine()) != null){
            String[] dataFields = line.split(APData.COMMA);

            String identifier = dataFields[0];
            identifier = identifier.trim(); // Remove leading and trail...
...ing whitespace

            String description = dataFields[1];
            description = description.trim();

            /** Longitude/Latitude in decimal degrees, then elevation c...
...ast as a double */
            Double lon = Double.parseDouble(dataFields[3]);
            Double lat = Double.parseDouble(dataFields[2]);
```

```
            Double ele = Double.parseDouble(dataFields[4])*0.01D;   // ...
...Convert feet to FL for consistency
            GPoint3D<Double> pt = new GPoint3D<Double>(lon, lat, ele);

            if(identifier.startsWith(APData.K) && a.airportExists(ident...
...ifier.substring(1))){
                Airport p = a.getAirport(identifier.substring(1));
                a.addAirport(identifier, p);                    // If a...
...irport already exists create reference using "K" prefix
            } else{
                Airport p = new Airport(identifier, description, pt);
                a.addAirport(p);
            }

            dataFields = null;
        }

        line = null;

        // Add Canadian airports removing the C prefix: e.g. add CYWG a...
...s YWG
        Iterator<String> e = a.getAirportKeyIterator();
        HashMap<String,Airport> canadianAirportsToAdd = new HashMap<Str...
...ing,Airport>(
                Settings.hashMapDefaultInitialCapacity,
                Settings.hashMapLoadFactor);
        while(e.hasNext()){
            String identifier = e.next();
            if(identifier.startsWith(APData.C) && identifier.length() =...
...= 4){
                String identifier3 = identifier.substring(1,4);
                if(!a.airportExists(identifier3)){
                    Airport canAirport = a.getAirport(identifier);
                    canadianAirportsToAdd.put(identifier3,canAirport);
                }
            }
        }
        for(String id:canadianAirportsToAdd.keySet()){
            a.addAirport(id, canadianAirportsToAdd.get(id));
        }

        d.close();

        canadianAirportsToAdd = null;

        return this;
    }
```

```
    /** Reads sector information into data structure */
    public FAPioAirspaceReader readSectors(String fileName) throws IOEx...
...ception{
        /* Node information starts at field 4 */
        final int START_SECTOR_NODE_FIELD = 4;

        BufferedReader d = new BufferedReader(new FileReader(fileName))...
...;

        String line;
        while ((line = d.readLine()) != null){
            String[] dataFields = line.split(APData.COMMA);
            String secName = new String(dataFields[0]);

            // Check to see if not a low altitude terminal sector
            char testIfLow = secName.charAt(3);
            if(!Character.isDigit(testIfLow)) continue;

            int secId = Integer.parseInt(dataFields[1]);
            int secFloor = Integer.parseInt(dataFields[2]);
            int secCeiling = Integer.parseInt(dataFields[3]);

            ASectorModule sec = new ASectorModule(secName, secId, secFl...
...oor, secCeiling);
            a.addSectorModule(sec);

            // Read latitude/longitude pairs that define sector geometr...
...y
            for (int i = START_SECTOR_NODE_FIELD; i < dataFields.length...
...; i++){
                String latLon = dataFields[i];
                latLon = latLon.trim();     // Remove leading and trail...
...ing whitespace
                String[] spt = latLon.split(APData.SPACE);
                double lat = Double.parseDouble(spt[0]);
                double lon = Double.parseDouble(spt[1]);
                GPoint2D <Double> node = new GPoint2D<Double>(lon,lat);...
...                 sec.addNode(node);
            }

            dataFields = null;
        }
        d.close();

        line = null;
```

```
        ArrayList<String> analysisCenters = new ArrayList<String>(FAPio...
....writeCenters.length);
        for(int i=0; i<FAPio.writeCenters.length; i++){
            analysisCenters.add(FAPio.writeCenters[i]);
        }

        // Create indices for the sectors
        int sectorIndex = 0;
        Iterator<ACenter> e = a.getCenterValueIterator();
        while(e.hasNext()){
            ACenter center = e.next();
            String centerS = center.getName();
            Iterator<ASector> e2 = center.getSectorValueIterator();
            while(e2.hasNext()){
                ASector sector = e2.next();
                if(analysisCenters.contains(centerS)){
                    sector.setArrayIndex(sectorIndex);
                    sectorIndex++;
                }
            }
        }

        a.setNoSectors(sectorIndex);

        return this;
    }

    /** Read capacity value of sector, i.e. the Monitor Alert Parameter...
... (MAP). */
    public FAPioAirspaceReader readSectorMaps(String fileName) throws I...
...OException{
        BufferedReader d = new BufferedReader(new FileReader(fileName))...
...;

        String line;
        while ((line = d.readLine()) != null){
            String[] dataFields = line.split(APData.COMMA);

            String secName = new String(dataFields[0]);
            ACenter center = a.getCenter(secName.substring(0,3));

            if(!center.containsSector(secName)) continue;

            ASector sector = center.getSector(secName);

            int mapValue = Integer.parseInt(dataFields[1]);
```

```
            sector.setMapValue(mapValue);

            dataFields = null;

        }
        d.close();

        line = null;

        return this;
    }

    /* Read the NAVAIDs and Fixes that are contained in each sector */
    public FAPioAirspaceReader readNavaidsAndFixesInSector(String navai...
...dFileName,
            String fixFileName) throws IOException{

        for(int i=0; i<=1; i++){
            BufferedReader d = null;
            if(i == 0){
                // First read NAVAIDs
                d = new BufferedReader(new FileReader(navaidFileName));...
...            }else{
                // then read fixes (file format same for both)
                d = new BufferedReader(new FileReader(fixFileName));
            }

            String line;
            while((line = d.readLine()) != null){
                int firstComma = line.indexOf(APData.COMMA);
                int secondComma = line.indexOf(APData.COMMA, firstComma...
...+1);
                int thirdComma = line.indexOf(APData.COMMA, secondComma...
...+1);
                int fourthComma = line.indexOf(APData.COMMA, thirdComma...
...+1);


                //String[] dataFields = line.split(APData.COMMA);

                // Read data from file
                String identifier = line.substring(0, firstComma);
                ACenter center = a.getCenter(line.substring(firstComma+...
...1, secondComma));
                //String identifier = dataFields[0];
                //ACenter center = a.getCenter(dataFields[1]);
```

```
                //String secName = dataFields[2];
                String secName = line.substring(secondComma+1, thirdCom...
...ma);
                if(!center.containsSector(secName)) continue;

                ASector sector = center.getSector(secName);
                double floor = Double.parseDouble(line.substring(thirdC...
...omma+1, fourthComma));
                double ceiling = Double.parseDouble(line.substring(four...
...thComma+1, line.length()));
                //double floor = Double.parseDouble(dataFields[3]);
                //double ceiling = Double.parseDouble(dataFields[4]);

                if(i == 0){
                    // Add navaid info to sector
                    sector.addNavaidToSector(identifier, floor, ceiling...
...);
                }else{
                    // Add fix info to sector
                    sector.addFixToSector(identifier, floor, ceiling);
                }

                //dataFields = null;
                line = null;
            }

            d.close();
        }

        // Check for duplicates
        for(String centerS:FAPio.writeCenters){
            ACenter center = a.getCenter(centerS);
            Iterator<ASector> sectorIterator = center.getSectorValueIte...
...rator();
            while(sectorIterator.hasNext()){
                ASector sector = sectorIterator.next();
                sector.removeDuplicateNavaidsFixes();
            }
        }

        return this;
    }


    public FAPioAirspaceReader readAirwaySectorIntersections(
            String intersectionFileName) throws IOException{
```

```
        BufferedReader d = new BufferedReader(new FileReader(intersecti...
...onFileName));

        // Create an arraylist for checking if center is under consider...
...ation
        ArrayList<String> analysisCenters = new ArrayList<String>(FAPio...
....writeCenters.length);
        for(int i=0; i<FAPio.writeCenters.length; i++){
            analysisCenters.add(FAPio.writeCenters[i]);
        }

        String line;
        int[] commaLocations = new int[11];
        while((line = d.readLine()) != null){
            for(int i=0; i<commaLocations.length; i++){
                if(i == 0){
                    commaLocations[i] = line.indexOf(APData.COMMA);
                }else{
                    commaLocations[i] = line.indexOf(APData.COMMA, comm...
...aLocations[i-1]+1);
                }
            }

            //String[] dataFields = line.split(APData.COMMA);

            // Check that the center is under analysis (field index 3)
            String centerS = line.substring(commaLocations[2]+1, commaL...
...ocations[3]);
            if(!analysisCenters.contains(centerS)){
                continue;
            }
            ACenter center = a.getCenter(centerS);

            // Fields
            String airwayS = line.substring(0, commaLocations[0]); //da...
...taFields[0];
            String fromId = line.substring(commaLocations[0]+1, commaLo...
...cations[1]); //dataFields[1];
            String toId = line.substring(commaLocations[1]+1, commaLoca...
...tions[2]); //dataFields[2];
            String secName = line.substring(commaLocations[3]+1, commaL...
...ocations[4]); //dataFields[4];

            if(!center.containsSector(secName)) continue;
            ASector sector = center.getSector(secName);
            //String module = dataFields[5];
            //String pt = dataFields[6];
```

```
            //String floor = dataFields[7];
            //String ceiling = dataFields[8];
            double latitude = Double.parseDouble(line.substring(commaLo...
...cations[8]+1, commaLocations[9]));
            double longitude = Double.parseDouble(line.substring(commaL...
...ocations[9]+1, commaLocations[10]));
            double elevation = Double.parseDouble(line.substring(commaL...
...ocations[10]+1, line.length()));

            // Add the intersection point to the sector
            sector.addAirwayIntersection(fromId, toId, longitude, latit...
...ude,
                    elevation);

            //dataFields = null;
        }
        d.close();

        line = null;
        analysisCenters = null;

        return this;
    }

    public FAPioAirspaceReader readArrivalsDeparturesForFlightFilter() ...
...throws IOException{
        BufferedReader d = new BufferedReader(new FileReader(Settings.e...
...tmsArrivalFile));
        // Read arrivals
        ArrayList<String> arrivalFilter = new ArrayList<String>();
        String line;
        while((line = d.readLine()) != null){
            String[] dataFields = line.split(APData.COMMA);

            if(dataFields.length < 11) continue;

            String flightCallsign = dataFields[6];
            int arriveTime = Integer.parseInt(dataFields[4].substring(0...
...,2));

            if(arriveTime <= Settings.arrivalTimeFilter)
                a.addFlightToFlightFilter(flightCallsign);
        }

        d.close();

        d = new BufferedReader(new FileReader(Settings.etmsDepartureFil...
```

```
...e));

        // Read departures
        while((line = d.readLine()) != null){
            String[] dataFields = line.split(APData.COMMA);

            if(dataFields.length < 17) continue;

            String flightCallsign = dataFields[6];
            int departTime = Integer.parseInt(dataFields[4].substring(0...
...,2));

            if(departTime >= Settings.departTimeFilter)
                a.addFlightToFlightFilter(flightCallsign);

            String departTimeString = dataFields[4];

            // Add this departure time to the list of departure times b...
...y flight
            // callsign.
            if(departTimeString.length() == 6){
                a.addFlightDepartureTime(flightCallsign, departTimeStri...
...ng);
            }
        }

        return this;
    }



}
```

FAPioAirspaceWriter.java

```
/*
 * FAPioAirspaceWriter.java
 *
 * Created on May 28, 2007, 12:57 AM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author Jeff
 */
import java.util.*;
import java.io.*;
public class FAPioAirspaceWriter {
    private final Airspace a;

    /** Creates a new instance of FAPioAirspaceWriter */
    public FAPioAirspaceWriter(Airspace airspace) {
        a = airspace;
    }

    /* Writes a copy of the airways for verification purposes */
    public FAPioAirspaceWriter writeAirwaysCopy(String fileName) throws...
... IOException{
        PrintWriter out = new PrintWriter(new BufferedWriter(new FileWr...
...iter(fileName + ".cpy")));

        ArrayList<Iterator<String>> eList = new ArrayList<Iterator<Stri...
...ng>>(3);
        eList.add(a.getJetRouteIterator());
        eList.add(a.getVictorAirwayIterator());
        eList.add(a.getRnavRouteIterator());

        for(Iterator<String> e:eList){

            while(e.hasNext()){
                String s = e.next();
                Airway awy = a.getAirway(s);
                for (int j = 0; j < awy.noPoints(); j++){
                    GPoint2D<Double> pt = awy.getPoint(j);
                    String ptDesignator = awy.getPointIdentifier(j);
                    out.print(s);
                    out.print(',');
                    out.print(ptDesignator);
```

```
                    out.print(',');
                    out.print(j);
                    out.print(',');
                    out.print(pt.getLatitude());
                    out.print(',');
                    out.println(pt.getLongitude());
                }
            }

        }
        out.close();

        return this;
    }


    /** Writes a copy of the navigational fix data for verification pur...
...poses */
    public FAPioAirspaceWriter writeFixesCopy(String fileName) throws I...
...OException{
        PrintWriter out = new PrintWriter(new BufferedWriter(new FileWr...
...iter(fileName + ".cpy")));

        Iterator<String> e = a.getFixIterator();
        while(e.hasNext()){
            String identifier = e.next();
            GPoint2D<Double> pt = a.getFix(identifier);
            out.print(identifier);
            out.print(APData.SPACE);
            out.print(pt.getLatitude());
            out.print(APData.SPACE);
            out.print(pt.getLongitude());
            out.print(APData.SPACE);
            out.println(a.getFixType(identifier));
        }
        out.close();

        return this;
    }




    /** Writes a copy of the NAVAID data for verification purposes */
    public FAPioAirspaceWriter writeNavaidsCopy(String fileName) throws...
... IOException{
        PrintWriter out = new PrintWriter(new BufferedWriter(new FileWr...
```

```
...iter(fileName + ".cpy")));

        Iterator<String> e = a.getNavaidIterator();
        while(e.hasNext()){
            String identifier = e.next();
            GPoint2D<Double> pt = a.getNavaid(identifier);
            out.print(identifier);
            out.print(APData.SPACE);
            out.print(pt.getLatitude());
            out.print(APData.SPACE);
            out.println(pt.getLongitude());
        }
        out.close();

        return this;
    }



    /** Write a copy of the airport information for verification purpos...
...es */
    public FAPioAirspaceWriter writeAirportsCopy(String fileName) throw...
...s IOException{
        PrintWriter out = new PrintWriter(new BufferedWriter(new FileWr...
...iter(fileName + ".cpy")));

        Iterator<String> e = a.getAirportKeyIterator();          // Nee...
...d to use the keys since some US airports have both
        while(e.hasNext()){                              // 3-Letter an...
...d "K" identifiers
            String identifier = e.next();
            Airport p = a.getAirport(identifier);

            out.print(identifier);
            out.print(',');
            out.print(p.getDescription());
            out.print(',');
            out.print(p.getLatitude());
            out.print(',');
            out.print(p.getLongitude());
            out.print(',');
            out.print(p.getElevation());
            out.println(' ');
        }
        out.close();

        return this;
```

```
    }



    /** Write a copy of the sector information for verification purpose...
...s */
    public FAPioAirspaceWriter writeSectorsCopy(String fileName) throws...
... IOException{
        PrintWriter out = new PrintWriter(new BufferedWriter(new FileWr...
...iter(fileName + ".cpy")));

        Iterator<ACenter> e = a.getCenterValueIterator();
        while(e.hasNext()){
            ACenter c = e.next();
            Iterator<ASector> eSec = c.getSectorValueIterator();

            while(eSec.hasNext()){
                ASector sec = eSec.next();
                for (int j=0; j<sec.noModules(); j++){
                    ASectorModule secMod = sec.getModule(j);

                    // Write a copy of the sector module details for ve...
...rification
                    out.print(secMod.getName());
                    out.print(' ');
                    out.print(secMod.getId());
                    out.print(' ');
                    out.print(secMod.getFloor());
                    out.print(' ');
                    out.print(secMod.getCeiling());
                    for (int i = 0; i < secMod.noNodes(); i++){
                        out.print(' ');
                        GPoint2D node = secMod.getNode(i);
                        out.print(node.getLatitude());
                        out.print(' ');
                        out.print(node.getLongitude());
                    }
                    out.println(' ');
                }
            }
        }
        out.close();

        return this;
    }

}
```

FAPioApcdmInputWriter.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author Jeff
 */
import java.util.*;
import java.util.concurrent.*;
import java.io.*;
import java.text.*;
public class FAPioApcdmInputWriter {
    public class SectorDemand{
        public int clusterIndex;
        public int pathIndex;
        public int sectorIndex;
        public double startTime;
        public double endTime;
        public int startTimeInt;
        public int endTimeInt;

        public SectorDemand(){

        }

        public SectorDemand(int clusterIndex_, int pathIndex_, int sect...
...orIndex_,
                double startTime_, double endTime_){
            clusterIndex = clusterIndex_;
            pathIndex = pathIndex_;
            sectorIndex = sectorIndex_;
            startTime = startTime_;
            endTime = endTime_;

            startTimeInt = (int)Math.round(startTime);
            endTimeInt = (int)Math.round(endTime);
        }
    }


    Scenario flightPlanScenario_;
    ArrayList<ASector> constrainedSectors_;

    private ArrayList<SectorDemand> summarizeSectorOccupancy(){
```

```
        ArrayList<SectorDemand> sectorDemandList = new ArrayList<Sector...
...Demand>();

        // Get a copy of the constrained sectors
        constrainedSectors_ = APData.network.flowConstrainedSectors();

        // Get a copy of the routing results
        ConcurrentHashMap<Integer, ArrayList<ArrayList<Link3D>>>
                kShortestPathsByCluster = flightPlanScenario_.KShortest...
...PathsByCluster();

        // Iterate through each of the clusters
        for(int clusterIndex = 1; clusterIndex <= kShortestPathsByClust...
...er.keySet().size(); clusterIndex++){

            // Get the departure time for flights in this cluster
            ArrayList<Double> flightDepartureTimes = new ArrayList<Doub...
...le>();
            ArrayList<Double> flightDepartureTimesCopy = new ArrayList<...
...Double>();

            ArrayList<String> flightIdsInCluster = flightPlanScenario_....
...getFlightsInCluster(clusterIndex);
            for(String flightId:flightIdsInCluster){
                Flight f = APData.airspace.getFlight(flightId);

                double departureTime = f.getDepartureTimeMinutes();

                flightDepartureTimes.add(departureTime);
            }

            // Iterate through each of the paths
            ArrayList<ArrayList<Link3D>> pathLinks = kShortestPathsByCl...
...uster.get(clusterIndex);


            // Use a label index of 11 for flights > 18 kft
            // Use a label index of 10 for flights < 18 kft cruising al...
...titude
            int labelIndex = 11;
            if (clusterIndex <= Settings.numberFlightClustersLT18kft) {...
...                 labelIndex = 10;
            }

            for(int i=0; i<pathLinks.size(); i++){
                // Get a copy of the departure times
                flightDepartureTimesCopy = new ArrayList<Double>(flight...
```

```
...DepartureTimes.size());
                for(int j=0; j<flightDepartureTimes.size(); j++){
                    flightDepartureTimesCopy.add(flightDepartureTimes.g...
...et(j));
                }

                ArrayList<Link3D> links = pathLinks.get(i);
                for(int j=0; j<links.size(); j++){
                    Link3D link = links.get(j);
                    double linkTravelTime = link.getDoubleValue(labelIn...
...dex);

                    ASector sector = link.getParentSector();
                    int sectorIndex = sector.getArrayIndex();

                    // Update the time
                    for(int k=0; k<flightDepartureTimesCopy.size(); k++...
...){
                        double startTime = flightDepartureTimesCopy.get...
...(k);
                        double endTime = startTime + linkTravelTime;

                        flightDepartureTimesCopy.set(k, endTime);

                        // Add to demand if a constrained sector
                        if(constrainedSectors_.contains(sector)){
                            SectorDemand sectorDemand = new SectorDeman...
...d(clusterIndex,
                                    i, sectorIndex, startTime, endTime)...
...;

                            sectorDemandList.add(sectorDemand);
                        }
                    }

                }
            }


        }

        return sectorDemandList;
    }

    public ArrayList<Double> constrainedSectorCapacities(){
        ArrayList<Double> capacities = new ArrayList<Double>(constraine...
...dSectors_.size());
```

```
        // Forecasted weather is the first scenario
        Scenario wxForecastScenario = APData.network.getWeatherScenario...
...(0);

        // Iterate through each of the constrained sectors
        for(int i=0; i<constrainedSectors_.size(); i++){
            ASector sector = constrainedSectors_.get(i);
            int sectorIndex = sector.getArrayIndex();

            // Get the capacity at the start of the forecasted weather ...
...event
            int startTime = (Settings.analysisTimeFilterStart + 1)*4;

            capacities.add(wxForecastScenario.getSectorCapacity15Min(se...
...ctorIndex, startTime));
        }

        return capacities;
    }

    /** Create the integer programming input file. */
    public FAPioApcdmInputWriter writeApcdmInputFile(){

        // Summarize sector occupancy for sector demand analysis
        ArrayList<SectorDemand> sectorDemandList = summarizeSectorOccup...
...ancy();

        // Get the capacity for each of the constrained sectors
        ArrayList<Double> sectorCapacities = constrainedSectorCapacitie...
...s();

        // Find the maximum sector capacity
        double maxCapacityDouble = 0.0D;
        for(int i=0; i<sectorCapacities.size(); i++){
            maxCapacityDouble = Math.max(maxCapacityDouble, sectorCapac...
...ities.get(i));
        }
        int maxCapacity = (int)Math.ceil(maxCapacityDouble);


        // Some model parameters (suggested from APCDM Transportation S...
...cience Paper 2006)
        final double gamma_s = 0.361 * 60.0;
        ArrayList<Double> mu_sn = new ArrayList<Double>(maxCapacity);
        for(int n=0; n<= maxCapacity; n++){
            mu_sn.add((gamma_s/5)*n*n);
```

```
        }


        // Create the formatter for double output
        DecimalFormat decimalFormat = new DecimalFormat("#0.0000");


        try{
            PrintWriter out = new PrintWriter(new BufferedWriter(
                new FileWriter(Settings.apcdmFile)));

            // Create the header lines
            out.println("\\Problem name: apcdm.lp");
            out.println("");
            out.println("Minimize");
            out.print(" obj: ");

            ////////////////////////////////////////////////////////...
...////////
            // 61(a) : Add first objective function term (c_fp * x_fp)
            ConcurrentHashMap<Integer, ArrayList<ArrayList<Link3D>>>
                    kShortestPathsByCluster = flightPlanScenario_.KShor...
...testPathsByCluster();

            // Maximum of 5 terms per line
            int newLineCnt = 0;

            for(int clusterIndex = 1; clusterIndex <= kShortestPathsByC...
...luster.keySet().size(); clusterIndex++){
                // Get the number of flights in this cluster
                ArrayList<String> flightIdsInCluster = flightPlanScenar...
...io_.getFlightsInCluster(clusterIndex);
                int numFlightsInCluster = flightIdsInCluster.size();

                // Use a label index of 11 for flights > 18 kft
                // Use a label index of 10 for flights < 18 kft cruisin...
...g altitude
                int labelIndex = 11;
                if(clusterIndex <= Settings.numberFlightClustersLT18kft...
...)
                    labelIndex = 10;

                ArrayList<ArrayList<Link3D>> pathLinks = kShortestPaths...
...ByCluster.get(clusterIndex);

                for(int i=0; i<pathLinks.size(); i++){
                    double clusterKTravelTime = 0.0D;
```

```
                    for(Link3D link:pathLinks.get(i)){
                        clusterKTravelTime += link.getDoubleValue(label...
...Index);
                    }
                    if(Double.isNaN(clusterKTravelTime) ||
                            clusterKTravelTime == Double.MAX_VALUE ||
                            clusterKTravelTime == Double.POSITIVE_INFIN...
...ITY ||
                            clusterKTravelTime == Double.NEGATIVE_INFIN...
...ITY)
                        continue;

                    String travelTimeS = decimalFormat.format(clusterKT...
...ravelTime);

                    // Check for bad values
                    if(travelTimeS.contains("?")) continue;

                    // Multiply travel time by number of flights in clu...
...ster
                    clusterKTravelTime *= numFlightsInCluster;

                    // Create the input variable
                    String varName = "CL" + Integer.toString(clusterInd...
...ex) + "K" +
                            Integer.toString(i);

                    String outS = travelTimeS + " " +
                            varName + " + ";

                    // Print the cost term
                    out.print(outS);

                    if(newLineCnt++ > 5){
                        out.println("");
                        newLineCnt = 0;
                    }
                }

                // Print the cancellation surrogate
                String cancellationSurrogate = "CL" + Integer.toString(...
...clusterIndex) + "X";
                out.print("9999999 ");
                out.print(cancellationSurrogate);
                out.print(" + ");
            }
            ////////////////////////////////////////////////////////...
```

```
.../////////


            ////////////////////////////////////////////////////////...
.../////////
            // 61(a) : Add second objective function term (mu_sn * y_sn...
...)
            for(int i=0; i<constrainedSectors_.size(); i++){
                newLineCnt = 0;

                ASector sector = constrainedSectors_.get(i);
                int sectorIndex = sector.getArrayIndex();

                for(int n = 1; n <= maxCapacity; n++){
                    out.print(decimalFormat.format(mu_sn.get(n)));

                    out.print(" ");

                    out.print("y");
                    out.print(sectorIndex);
                    out.print("n");
                    out.print(n);

                    out.print(" + ");


                    if(newLineCnt++ > 5){
                        out.println("");
                        newLineCnt = 0;
                    }
                }
            }


            ////////////////////////////////////////////////////////...
.../////////
            // 61(a) : Add sixth objective function term (gamma_s * w_s...
...)
            for(int i=0; i<constrainedSectors_.size(); i++){
                newLineCnt = 0;

                ASector sector = constrainedSectors_.get(i);
                int sectorIndex = sector.getArrayIndex();

                out.print(" ");
                out.print(decimalFormat.format(gamma_s));
                out.print(" ");
```

```
                out.print("w");
                out.print(sectorIndex);

                if(i < constrainedSectors_.size() - 1){
                    out.println(" + ");
                }
            }


            ////////////////////////////////////////////////////////...
.../////////
            // Add the header lines for the constraints
            out.println("");
            out.println("Subject To");

            // Counter for constraints
            int constraintCount = 1;


            ////////////////////////////////////////////////////////...
.../////////
            // 61(b) : First constraint
            for(int clusterIndex = 1; clusterIndex <= kShortestPathsByC...
...luster.keySet().size(); clusterIndex++){

                ArrayList<ArrayList<Link3D>> pathLinks = kShortestPaths...
...ByCluster.get(clusterIndex);

                for(int i=0; i<pathLinks.size(); i++){
                    if(i == 0){
                        out.print(" c");
                        out.print(constraintCount);

                        // Increment the constraint counter
                        constraintCount++;

                        out.print(": ");
                    }

                    // Create the input variable
                    String varName = "CL" + Integer.toString(clusterInd...
...ex) + "K" +
                            Integer.toString(i);

                    out.print(varName);

                    out.print(" + ");
```

```
                    if(i == pathLinks.size() - 1){
                        // Add the cancellation surrogate
                        String cancellationSurrogate = "CL" + Integer.t...
...oString(clusterIndex) + "X";
                        out.print(cancellationSurrogate);

                        out.println(" = 1");
                    }
                }
            }


            ////////////////////////////////////////////////////////...
.../////////
            // 61(c) : Second constraint
            int startTimeMin = (Settings.analysisTimeFilterStart + 1)*6...
...0;
            int endTimeMin   = (Settings.analysisTimeFilterEnd   + 2)*6...
...0;
            //newLineCnt = 0;
            // Iterate through each of the constrained sectors
            for(int i=0; i<constrainedSectors_.size(); i++){
                ASector sector = constrainedSectors_.get(i);
                int sectorIndex = sector.getArrayIndex();

                // Iterate through each of the time periods
                for(int t=startTimeMin; t<=endTimeMin; t++){
                    newLineCnt = 0;
                    out.print(" c");
                    out.print(constraintCount);

                    // Increment the constraint counter
                    constraintCount++;

                    out.print(": ");

                    // Add the variable representing peak sector occupa...
...ncy count
                    out.print("n");
                    out.print(sectorIndex);
                    out.print(" ");

                    HashMap<Integer,Integer> alreadyPrinted = new HashM...
...ap<Integer,Integer>();
                    for(int j=0; j<sectorDemandList.size(); j++){
                        SectorDemand sectorDemand = sectorDemandList.ge...
```

```
...t(j);

                        if(sectorDemand.sectorIndex == sectorIndex &&
                                sectorDemand.startTimeInt <= t &&
                                sectorDemand.endTimeInt >= t){
                            // Check if a variable has already been pri...
...nted
                            if(alreadyPrinted.containsKey(sectorDemand....
...clusterIndex) &&
                                    alreadyPrinted.get(sectorDemand.clu...
...sterIndex) ==
                                    sectorDemand.pathIndex) continue;

                            // Store the indices to ensure no duplicate...
...s
                            alreadyPrinted.put(sectorDemand.clusterInde...
...x,
                                    sectorDemand.pathIndex);

                            out.print(" - ");

                            // Create the variable string
                            String varName = "CL" + Integer.toString(se...
...ctorDemand.clusterIndex)
                                    + "K" + Integer.toString(sectorDema...
...nd.pathIndex);

                            out.print(varName);

                            if(newLineCnt++ > 5){
                                out.println("");
                                newLineCnt = 0;
                            }
                        }
                    }

                    // Right hand side of the constraint
                    out.println(" >= 0");
                }
            }


            ////////////////////////////////////////////////////////...
.../////////
            // 61(d) : Third constraint
            //newLineCnt = 0;
            // Iterate through each of the constrained sectors
```

```
            final double H_INVERSE = 1/60.0D;
            for(int i=0; i<constrainedSectors_.size(); i++){
                newLineCnt = 0;

                ASector sector = constrainedSectors_.get(i);
                int sectorIndex = sector.getArrayIndex();

                out.print(" c");
                out.print(constraintCount);

                // Increment the constraint counter
                constraintCount++;

                out.print(": ");

                // Add the variable representing average sector occupan...
...cy count
                out.print("w");
                out.print(sectorIndex);
                out.print(" ");

                HashMap<Integer, Integer> alreadyPrinted = new HashMap<...
...Integer, Integer>();
                for (int j = 0; j < sectorDemandList.size(); j++) {
                    SectorDemand sectorDemand = sectorDemandList.get(j)...
...;
                    if (sectorDemand.sectorIndex == sectorIndex &&
                            sectorDemand.startTimeInt >= startTimeMin &...
...&
                            sectorDemand.startTimeInt <= endTimeMin) {
                        // Check if a variable has already been printed...
...                        if (alreadyPrinted.containsKey(sectorDemand.clu...
...sterIndex) &&
                                alreadyPrinted.get(sectorDemand.cluster...
...Index) ==
                                sectorDemand.pathIndex) {
                            continue;
                        }
                        // Store the indices to ensure no duplicates
                        alreadyPrinted.put(sectorDemand.clusterIndex,
                                sectorDemand.pathIndex);

                        double coefficient = H_INVERSE * (sectorDemand....
...endTime
                                - sectorDemand.startTime);

                        out.print(" - ");
```

```
                        out.print(decimalFormat.format(coefficient));
                        out.print(" ");

                        // Create the input variable
                        String varName = "CL" + Integer.toString(sector...
...Demand.clusterIndex)
                                + "K" +
                            Integer.toString(sectorDemand.pathIndex);
                        out.print(varName);

                        if(newLineCnt++ > 5){
                            out.println("");
                            newLineCnt = 0;
                        }
                    }

                }

                // Right hand side of the constraint
                out.println(" = 0");
            }



            ////////////////////////////////////////////////////////...
...//////////
            // 61(e) : Fourth constraint
            // Iterate through each of the constrained sectors
            for(int i=0; i<constrainedSectors_.size(); i++){
                newLineCnt = 0;

                ASector sector = constrainedSectors_.get(i);
                int sectorIndex = sector.getArrayIndex();

                out.print(" c");
                out.print(constraintCount);

                // Increment the constraint counter
                constraintCount++;

                out.print(": ");

                // Add the variable for peak sector occupancy count
                out.print("n");
                out.print(sectorIndex);
                out.print(" - ");
```

```
                // Add the variable for average sector occupancy count
                out.print("w");
                out.print(sectorIndex);

                for(int n = 1; n <= maxCapacity; n++){
                    out.print(" - ");

                    out.print(n);
                    out.print(" ");
                    out.print("y");
                    out.print(sectorIndex);
                    out.print("n");
                    out.print(n);

                    if(newLineCnt++ > 5){
                        out.println("");
                        newLineCnt = 0;
                    }
                }

                // Print the right hand side
                out.println(" = 0");
            }



            ////////////////////////////////////////////////////////...
...//////////
            // 61(f) : Fifth constraint
            // Iterate through each of the constrained sectors
            for(int i=0; i<constrainedSectors_.size(); i++){
                newLineCnt = 0;

                ASector sector = constrainedSectors_.get(i);
                int sectorIndex = sector.getArrayIndex();

                out.print(" c");
                out.print(constraintCount);

                // Increment the constraint counter
                constraintCount++;

                out.print(": ");

                for(int n = 0; n <= maxCapacity; n++){
                    out.print("y");
                    out.print(sectorIndex);
```

```
                out.print("n");
                out.print(n);

                if(n < maxCapacity){
                    out.print(" + ");
                }

                if(newLineCnt++ > 5){
                    out.println("");
                    newLineCnt = 0;
                }
            }

            // Print the right hand side
            out.println(" = 1");
        }


        ////////////////////////////////////////////////////////////...
...//////////
        // Add the header lines for the bounds
        out.println("");
        out.println("Bounds");

        // Add the sector capacities to the bounds
        for(int i=0; i<constrainedSectors_.size(); i++){
            ASector sector = constrainedSectors_.get(i);
            int sectorIndex = sector.getArrayIndex();

            double sectorCapacity = sectorCapacities.get(i);

            // Use a lower bound of zero
            out.print("0 <= n");

            out.print(sectorIndex);

            // The upper bound is the sector capacity
            out.print(" <= ");
            out.println(decimalFormat.format(sectorCapacity));
        }


        ////////////////////////////////////////////////////////////...
...//////////
        // Add the head lines for the binary decision variables
        out.println("");
        out.println("Binaries");
```

```
            ////////////////////////////////////////////////////////...
...//////////
            // Specify the binary decision variables
            for(int clusterIndex = 1; clusterIndex <= kShortestPathsByC...
...luster.keySet().size(); clusterIndex++){

                ArrayList<ArrayList<Link3D>> pathLinks = kShortestPaths...
...ByCluster.get(clusterIndex);

                for(int i=0; i<pathLinks.size(); i++){
                    out.print(" ");

                    // Create the input variable
                    String varName = "CL" + Integer.toString(clusterInd...
...ex) + "K" +
                            Integer.toString(i);

                    out.print(varName);

                    if(i == pathLinks.size() - 1){
                        out.println("");
                    }
                }
            }


            ////////////////////////////////////////////////////////...
...//////////
            // Add end of the document
            out.println("");
            out.println("End");

            out.close();
        }catch(IOException e){
            System.out.println("Could not create APCDM input file!");
        }

        return this;
    }

    public FAPioApcdmInputWriter(Scenario flightPlanScenario){
        flightPlanScenario_ = flightPlanScenario;

        writeApcdmInputFile();
```

```
        writeClusterPathFlightIdForApcdmOutput();
    }

    /** Create a listing of flights in each cluster to process the IP o...
...utput. */
    public FAPioApcdmInputWriter writeClusterPathFlightIdForApcdmOutput...
...(){
        ConcurrentHashMap<Integer, ArrayList<ArrayList<Link3D>>>
                    kShortestPathsByCluster = flightPlanScenario_.KShor...
...testPathsByCluster();

        int numClusters = kShortestPathsByCluster.keySet().size();

        try{
            PrintWriter out = new PrintWriter(new BufferedWriter(
                new FileWriter(Settings.apcdmFlightsInClusterFile)));

            out.println("Varname,ClusterIndex,KIndex,FlightId");

            for(int clusterIndex = 1; clusterIndex <= numClusters; clus...
...terIndex++){
                ArrayList<ArrayList<Link3D>> pathLinks = kShortestPaths...
...ByCluster.get(clusterIndex);

                // Get a listing of the flight ids for this cluster
                ArrayList<String> flightsInCluster = flightPlanScenario...
..._.getFlightsInCluster(clusterIndex);

                for (int i = 0; i < pathLinks.size(); i++) {
                    // Create the input variable
                    String varName = "CL" + Integer.toString(clusterInd...
...ex) + "K" +
                            Integer.toString(i);
                    for (int j = 0; j < flightsInCluster.size(); j++) {...
...                        // Print the variable name
                        out.print(varName);
                        out.print(",");
                        // Print the cluster index
                        out.print(clusterIndex);
                        out.print(",");
                        // Print the k-index
                        out.print(i);
                        out.print(",");

                        // Print the flight id
                        out.println(flightsInCluster.get(j));
                    }
```

```
                }

            }

            out.close();
        }catch(IOException e){
            System.out.println("Could not create listing of APCDM varia...
...bles file!");
        }

        return this;
    }
}
```

FAPioKmlWriter.java

```
/*
 * FAPioKmlWriter.java
 *
 * Created on June 7, 2007, 8:35 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author Jeff
 */
import java.util.*;
import java.util.concurrent.*;
import java.util.zip.*;
import java.io.*;
public class FAPioKmlWriter {
    // Region viewing parameters
    private static final String MINLODPIXELS = "512";
    private static final String MAXLODPIXELS = "-1";
    private static final String MINFADEEXTENT = "512";
    private static final String MAXFADEEXTENT = "512";

    private static final double FLIGHTLEVELTOALTITUDE = 100.0D;

    // Do not write external Navaids and Fixes for now
    private static final boolean WRITEEXTERNALNODES = false;

    // Write external airways and jet routes
    private static final boolean WRITEEXTERNALAIRWAYS = true;

    /** Creates a new instance of FAPioKmlWriter */
    public FAPioKmlWriter() {
    }

    public FAPioKmlWriter writeKmlFile(){
        Network n = APData.network;
        Airspace a = APData.airspace;

        try{
            PrintWriter out = new PrintWriter(new BufferedWriter(new Fi...
...leWriter(Settings.kmlFile)));

            addKmlDocumentStart(out);
//            addAirportStyleToKml(out);
```

```
//            addNavaidStyleToKml(out);
//            addFixStyleToKml(out);
//            addStarStyleToKml(out);
//            addSectorStyleToKml(out);
            addLinkStyleToKml(out);

            //addAirportsToKml(out,n);
            //addNavaidsToKml(out,n,a);
            //addFixesToKml(out,n,a);
            //addStarsToKml(out,n,a);

//            addSectorsToKml(out,n,a);
//            addLinksToKml(out,n,a);
            addPathLinksToKml(out, n, a);
            addPlannedLinksToKml(out, n, a);
            addIncrementalToKml(out, n, a);

            addKmlDocumentEnd(out);
            out.close();

        }catch(IOException e){
            System.out.println("Could not open kml file");
        }

        return this;
    }


    private FAPioKmlWriter addKmlDocumentStart(PrintWriter s){
        s.append("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");
        s.append("<kml xmlns=\"http://earth.google.com/kml/2.1\">\n");
        s.append("\t<Document>\n");
        s.append("\t\t<name>AirPlanJ</name>\n");
        s.append("\t\t<open>1</open>\n");
        s.append("\t\t<description>Welcome to the Airspace Planning res...
...ults browser!</description>\n");
        return this;
    }

    private FAPioKmlWriter addKmlDocumentEnd(PrintWriter s){
        s.append("\t</Document>\n");
        s.append("</kml>\n");
        return this;
    }

    private FAPioKmlWriter addAirportStyleToKml(PrintWriter s){
        s.append("\t\t<Style id=\"airport\">\n");
```

```
        s.append("\t\t\t<IconStyle>\n");
        s.append("\t\t\t\t<scale>1.5</scale>\n");
        s.append("\t\t\t\t<Icon>\n");
        s.append("\t\t\t\t<href>http://maps.google.com/mapfiles/kml/pal...
...4/icon28.png</href>\n");
        s.append("\t\t\t\t</Icon>\n");
        s.append("\t\t\t</IconStyle>\n");
        s.append("\t\t\t<LabelStyle>\n");
        s.append("\t\t\t\t<scale>1.5</scale>\n");
        s.append("\t\t\t</LabelStyle>\n");
        s.append("\t\t</Style>\n");
        return this;
    }


    private FAPioKmlWriter addAirportsToKml(PrintWriter s, Network n){
        s.append("\t\t<Folder>\n");
        s.append("\t\t\t<name>Airports</name>\n");
        s.append("\t\t\t<description>Airports in scope of analysis.</de...
...scription>\n");

        this.appendPointPlacemarks(n, s, Node3D.NodeTypes.Airport, "air...
...port", ASector.NullSector);

        s.append("\t\t</Folder>\n");

        return this;
    }

    private FAPioKmlWriter addNavaidStyleToKml(PrintWriter s){
        s.append("\t\t<Style id=\"navaid\">\n");
        s.append("\t\t\t<IconStyle>\n");
        s.append("\t\t\t\t<scale>0.5</scale>\n");
        s.append("\t\t\t\t<Icon>\n");
        s.append("\t\t\t\t<href>http://maps.google.com/mapfiles/kml/pal...
...4/icon25.png</href>\n");
        s.append("\t\t\t\t</Icon>\n");
        s.append("\t\t\t</IconStyle>\n");
        s.append("\t\t\t<LabelStyle>\n");
        s.append("\t\t\t\t<scale>0.5</scale>\n");
        s.append("\t\t\t</LabelStyle>\n");
        s.append("\t\t</Style>\n");
        return this;
    }

    private FAPioKmlWriter addFixStyleToKml(PrintWriter s){
        s.append("\t\t<Style id=\"fix\">\n");
```

```
        s.append("\t\t\t<IconStyle>\n");
        s.append("\t\t\t\t<scale>0.5</scale>\n");
        s.append("\t\t\t\t<Icon>\n");
        s.append("\t\t\t\t<href>http://maps.google.com/mapfiles/kml/pal...
...4/icon25.png</href>\n");
        s.append("\t\t\t\t</Icon>\n");
        s.append("\t\t\t</IconStyle>\n");
        s.append("\t\t\t<LabelStyle>\n");
        s.append("\t\t\t\t<scale>0.5</scale>\n");
        s.append("\t\t\t</LabelStyle>\n");
        s.append("\t\t</Style>\n");
        return this;
    }

    private FAPioKmlWriter addStarStyleToKml(PrintWriter s){
        s.append("\t\t<Style id=\"star\">\n");
        s.append("\t\t\t<IconStyle>\n");
        s.append("\t\t\t\t<scale>0.5</scale>\n");
        s.append("\t\t\t\t<Icon>\n");
        s.append("\t\t\t\t<href>http://maps.google.com/mapfiles/kml/pal...
...4/icon25.png</href>\n");
        s.append("\t\t\t\t</Icon>\n");
        s.append("\t\t\t</IconStyle>\n");
        s.append("\t\t\t<LabelStyle>\n");
        s.append("\t\t\t\t<scale>0.5</scale>\n");
        s.append("\t\t\t</LabelStyle>\n");
        s.append("\t\t</Style>\n");
        return this;
    }

    private FAPioKmlWriter addSectorStyleToKml(PrintWriter s){
        s.append("\t\t<Style id=\"sector\">\n");
        s.append("\t\t\t<LineStyle>\n");
        s.append("\t\t\t\t<color>666aff52</color>\n");
        s.append("\t\t\t\t<width>2</width>\n");
        s.append("\t\t\t</LineStyle>\n");
        s.append("\t\t\t<PolyStyle>\n");
        s.append("\t\t\t\t<color>666aff52</color>\n");
        s.append("\t\t\t</PolyStyle>\n");
        s.append("\t\t</Style>\n");
        return this;
    }

    private FAPioKmlWriter addLinkStyleToKml(PrintWriter s){
        s.append("\t\t<Style id=\"link\">\n");
        s.append("\t\t\t<LineStyle>\n");
        s.append("\t\t\t\t<color>666aff52</color>\n");
```

```
        s.append("\t\t\t\t<width>4</width>\n");
        s.append("\t\t\t</LineStyle>\n");
        s.append("\t\t</Style>\n");
        return this;
    }

    private FAPioKmlWriter appendPointPlacemarks(Network n,PrintWriter ...
...s,
            Node3D.NodeTypes nodeType, String style, ASector sector){


        // Iterate through each object in network and write as a kml Pl...
...acemark
        Iterator<String> e = n.getNodeIterator();
        int i=0;
        while(e.hasNext()){
            if(i > 500) break;
            String identifierS = e.next();
            Node3D<Double> pt = n.getNode(identifierS);

            if(pt.getNodeType().equals(nodeType) && pt.getParentSector(...
...).equals(sector)){
                // Need to remove "_APT" suffix for airports
                if(pt.getNodeType().equals(Node3D.NodeTypes.Airport)){
                    identifierS = identifierS.split("_")[0];
                }

                s.append("\t\t\t<Placemark>\n");
                s.append("\t\t\t\t<name>" + identifierS + "</name>\n");...
...                s.append("\t\t\t\t<visibility>0</visibility>\n");
                s.append("\t\t\t\t<styleUrl>#" + style + "</styleUrl>\n...
...");

                // Define visible region
                Double northLatitude = pt.getLatitude() + 1.0D;
                Double southLatitude = pt.getLatitude() - 1.0D;
                Double eastLongitude = pt.getLongitude() + 1.0D;
                Double westLongitude = pt.getLongitude() - 1.0D;
                s.append("\t\t\t\t<Region>\n");
                s.append("\t\t\t\t  <LatLonAltBox>\n");
                s.append("\t\t\t\t    <north>" + northLatitude.toString...
...() + "</north>\n");
                s.append("\t\t\t\t    <south>" + southLatitude.toString...
...() + "</south>\n");
                s.append("\t\t\t\t    <east>" + eastLongitude.toString(...
...) + "</east>\n");
                s.append("\t\t\t\t    <west>" + westLongitude.toString(...
```

```
...) + "</west>\n");
                s.append("\t\t\t\t    <minAltitude>0</minAltitude>\n");...
...                s.append("\t\t\t\t    <maxAltitude>5000</maxAltitude>\n...
...");
                s.append("\t\t\t\t  </LatLonAltBox>\n");
                s.append("\t\t\t\t  <Lod>\n");
                s.append("\t\t\t\t    <minLodPixels>" + MINLODPIXELS + ...
..."</minLodPixels>\n");
                s.append("\t\t\t\t    <maxLodPixels>" + MAXLODPIXELS + ...
..."</maxLodPixels>\n");
                s.append("\t\t\t\t    <minFadeExtent>" + MINFADEEXTENT ...
...+ "</minFadeExtent>\n");
                s.append("\t\t\t\t    <maxFadeExtent>" + MAXFADEEXTENT ...
...+ "</maxFadeExtent>\n");
                s.append("\t\t\t\t  </Lod>\n");
                s.append("\t\t\t\t</Region>\n");

                // Point Geometry
                s.append("\t\t\t\t<Point>\n");

                // Elevation
                double elevation = pt.getElevation();

                // Use absolute altitude for navaids and fixes, clamp t...
...o ground
                // (default) for airports
                if(pt.getNodeType().equals(Node3D.NodeTypes.NavaidFloor...
...) ||
                        pt.getNodeType().equals(Node3D.NodeTypes.Navaid...
...Centroid) ||
                        pt.getNodeType().equals(Node3D.NodeTypes.Navaid...
...Ceiling)
                        || pt.getNodeType().equals(Node3D.NodeTypes.Fix...
...Floor)
                        || pt.getNodeType().equals(Node3D.NodeTypes.Fix...
...Centroid)
                        || pt.getNodeType().equals(Node3D.NodeTypes.Fix...
...Ceiling)
                        || pt.getNodeType().equals(Node3D.NodeTypes.Sta...
...r)){
                    s.append("\t\t\t\t<altitudeMode>absolute</altitudeM...
...ode>\n");
                    elevation *= FLIGHTLEVELTOALTITUDE;
                }
```

```
                s.append("\t\t\t\t<coordinates>"+pt.getLongitude().toSt...
...ring()+
                        ","+pt.getLatitude().toString()+","+
                        Double.toString(elevation)+"</coordinates>\n");...
...                s.append("\t\t\t\t</Point>\n");
                s.append("\t\t\t</Placemark>\n");

                //break;
            }
        }

        return this;


    }

    private FAPioKmlWriter addNavaidsToKml(PrintWriter s, Network n, Ai...
...rspace a){
        s.append("\t\t<Folder>\n");
        s.append("\t\t\t<name>Navaids</name>\n");
        s.append("\t\t\t<description>Navaids within extent of sectors u...
...sed in analysis.</description>\n");

        for(String centerS:FAPio.writeCenters){
            ACenter center = a.getCenter(centerS);
            s.append("\t\t\t<Folder>\n");
            s.append("\t\t\t<name>" + center.getName() + "</name>\n");
            Iterator<ASector> sectorIterator = center.getSectorValueIte...
...rator();
            while(sectorIterator.hasNext()){
                ASector sector = sectorIterator.next();
                s.append("\t\t\t<Folder>\n");
                s.append("\t\t\t<name>" + sector.getName() + "</name>\n...
...");
                this.appendPointPlacemarks(n, s, Node3D.NodeTypes.Navai...
...dFloor, "navaid", sector);
                this.appendPointPlacemarks(n, s, Node3D.NodeTypes.Navai...
...dCentroid, "navaid", sector);
                this.appendPointPlacemarks(n, s, Node3D.NodeTypes.Navai...
...dCeiling, "navaid", sector);
                s.append("\t\t\t</Folder>\n");
            }
            s.append("\t\t\t</Folder>\n");
        }

        if(WRITEEXTERNALNODES){
            // Write Navaids outside the defined centers
```

```
            s.append("\t\t\t<Folder>\n");
            s.append("\t\t\t<name> Other Navaids </name>\n");
            this.appendPointPlacemarks(n, s, Node3D.NodeTypes.NavaidCen...
...troid, "navaid", ASector.NullSector);
            s.append("\t\t\t</Folder>\n");
        }

        s.append("\t\t</Folder>\n");

        return this;
    }

    private FAPioKmlWriter addFixesToKml(PrintWriter s, Network n, Airs...
...pace a){
        s.append("\t\t<Folder>\n");
        s.append("\t\t\t<name>Fixes</name>\n");
        s.append("\t\t\t<description>Fixes within extent of sectors use...
...d in analysis.</description>\n");

        for(String centerS:FAPio.writeCenters){
            ACenter center = a.getCenter(centerS);
            s.append("\t\t\t<Folder>\n");
            s.append("\t\t\t<name>" + center.getName() + "</name>\n");
            Iterator<ASector> sectorIterator = center.getSectorValueIte...
...rator();
            while(sectorIterator.hasNext()){
                ASector sector = sectorIterator.next();
                s.append("\t\t\t<Folder>\n");
                s.append("\t\t\t<name>" + sector.getName() + "</name>\n...
...");
                this.appendPointPlacemarks(n, s, Node3D.NodeTypes.FixFl...
...oor, "fix", sector);
                this.appendPointPlacemarks(n, s, Node3D.NodeTypes.FixCe...
...ntroid, "fix", sector);
                this.appendPointPlacemarks(n, s, Node3D.NodeTypes.FixCe...
...iling, "fix", sector);
                s.append("\t\t\t</Folder>\n");
            }
            s.append("\t\t\t</Folder>\n");
        }

        if(WRITEEXTERNALNODES){
            // Write Fixes outside the defined centers
            s.append("\t\t\t<Folder>\n");
            s.append("\t\t\t<name> Other Fixes </name>\n");
            this.appendPointPlacemarks(n, s, Node3D.NodeTypes.FixCentro...
...id, "fix", ASector.NullSector);
```

```
            s.append("\t\t\t</Folder>\n");
        }


        s.append("\t\t</Folder>\n");

        return this;
    }

    private FAPioKmlWriter addStarsToKml(PrintWriter s, Network n, Airs...
...pace a){
        s.append("\t\t<Folder>\n");
        s.append("\t\t\t<name>Stars</name>\n");
        s.append("\t\t\t<description>Stars listed in flight plans.</des...
...cription>\n");

        // Stars are not defined for enroute sectors
        s.append("\t\t\t<Folder>\n");
        s.append("\t\t\t<name> All Stars </name>\n");
        this.appendPointPlacemarks(n, s, Node3D.NodeTypes.Star, "star",...
... ASector.NullSector);
        s.append("\t\t\t</Folder>\n");

        s.append("\t\t</Folder>\n");

        return this;
    }

    private FAPioKmlWriter addPlannedLinksToKml(PrintWriter s, Network ...
...n, Airspace a){
        Scenario flightPlanScenario = n.getFlightPlanScenario();
        ConcurrentHashMap<Integer, ArrayList<ArrayList<Link3D>>> kShort...
...estPathsByCluster =
                flightPlanScenario.KShortestPathsByCluster();

        s.append("\t\t<Folder>\n");
        s.append("\t\t\t<name>Flight Plan Paths</name>\n");
        s.append("\t\t\t<description>Planned paths for flights by clust...
...er.</description>\n");

        int numClusters = kShortestPathsByCluster.size();

        // Clusters are 1-based
        for(int i=1; i<=numClusters; i++){
            s.append("\t\t\t<Folder>\n");
            s.append("\t\t\t<name>Cluster " + Integer.toString(i) + "</...
...name>\n");
```

```
            ArrayList<String> flightsInCluster = flightPlanScenario.get...
...FlightsInCluster(i);

            for(int j=0; j<flightsInCluster.size(); j++){
                s.append("\t\t\t<Placemark>\n");
                s.append("\t\t\t<name>Path " + Integer.toString(j) + "<...
.../name>\n");
                s.append("\t\t\t<visibility>0</visibility>\n");
                s.append("\t\t\t\t<styleUrl>#link</styleUrl>\n");
                s.append("\t\t\t\t<Region>\n");
                s.append("\t\t\t\t  <Lod>\n");
                s.append("\t\t\t\t    <minLodPixels>" + MINLODPIXELS + ...
..."</minLodPixels>\n");
                s.append("\t\t\t\t    <maxLodPixels>" + MAXLODPIXELS + ...
..."</maxLodPixels>\n");
                s.append("\t\t\t\t    <minFadeExtent>" + MINFADEEXTENT ...
...+ "</minFadeExtent>\n");
                s.append("\t\t\t\t    <maxFadeExtent>" + MAXFADEEXTENT ...
...+ "</maxFadeExtent>\n");
                s.append("\t\t\t\t  </Lod>\n");
                s.append("\t\t\t\t</Region>\n");
                s.append("\t\t\t\t<MultiGeometry>\n");

                String flightId = flightsInCluster.get(j);

                for(int k=0; k<flightPlanScenario.noFlightLinks(flightI...
...d); k++){

                    Link3D pathLink = flightPlanScenario.getFlightLink(...
...flightId, k);

                    Node3D<Double> fromNode = pathLink.getFromNode();
                    Node3D<Double> toNode = pathLink.getToNode();

                    s.append("\t\t\t\t<LineString>\n");
                    s.append("\t\t\t\t<altitudeMode>absolute</altitudeM...
...ode>\n");
                    s.append("\t\t\t\t<coordinates>");
                    s.append(fromNode.getLongitude().toString() + ",");...
...                    s.append(fromNode.getLatitude().toString() + ",");
                    s.append(Double.toString(fromNode.getElevation()*10...
...0.0D)+ " ");
                    s.append(toNode.getLongitude().toString() + ",");
                    s.append(toNode.getLatitude().toString() + ",");
                    s.append(Double.toString(toNode.getElevation()*100....
...0D) + "\n");
```

```
                    s.append("\t\t\t\t</coordinates>\n");
                    s.append("\t\t\t\t</LineString>\n");
                }
                s.append("\t\t\t\t</MultiGeometry>\n");
                s.append("\t\t\t</Placemark>\n");
            }
            s.append("\t\t\t</Folder>\n");
        }
        s.append("\t\t</Folder>\n");

        return this;
    }

    /** Write incremental assignment results to the kml file. */
    private FAPioKmlWriter addIncrementalToKml(PrintWriter s, Network n...
..., Airspace a){
        int numWeatherScenarios = n.getNumberWeatherScenarios();
        ArrayList<Scenario> weatherScenarios = new ArrayList<Scenario>(...
...                 numWeatherScenarios);
        int numClusters = 0;
        for(int i=0; i<numWeatherScenarios; i++){
            Scenario wxScenario = n.getWeatherScenario(i);
            weatherScenarios.add(wxScenario);

            if(i == 0){
                ConcurrentHashMap<Integer, ArrayList<Link3D>> increment...
...alPaths =
                        wxScenario.IncrementalPathsByCluster();

                numClusters = incrementalPaths.size();
            }
        }

        s.append("\t\t<Folder>\n");
        s.append("\t\t\t<name>Incremental Assignment Results</name>\n")...
...;
        s.append("\t\t\t<description>Incremental assignment paths by cl...
...uster.</description>\n");

        // Clusters are 1-based
        for(int i=1; i<=numClusters; i++){
            s.append("\t\t\t<Folder>\n");
            s.append("\t\t\t<name>Cluster " + Integer.toString(i) + "</...
...name>\n");

            for(int j=0; j<numWeatherScenarios; j++){
                s.append("\t\t\t<Placemark>\n");
```

```
                s.append("\t\t\t<name>Weather Scenario " + Integer.toSt...
...ring(j) + "</name>\n");
                s.append("\t\t\t<visibility>0</visibility>\n");
                s.append("\t\t\t\t<styleUrl>#link</styleUrl>\n");
                s.append("\t\t\t\t<Region>\n");
                s.append("\t\t\t\t  <Lod>\n");
                s.append("\t\t\t\t    <minLodPixels>" + MINLODPIXELS + ...
..."</minLodPixels>\n");
                s.append("\t\t\t\t    <maxLodPixels>" + MAXLODPIXELS + ...
..."</maxLodPixels>\n");
                s.append("\t\t\t\t    <minFadeExtent>" + MINFADEEXTENT ...
...+ "</minFadeExtent>\n");
                s.append("\t\t\t\t    <maxFadeExtent>" + MAXFADEEXTENT ...
...+ "</maxFadeExtent>\n");
                s.append("\t\t\t\t  </Lod>\n");
                s.append("\t\t\t\t</Region>\n");
                s.append("\t\t\t\t<MultiGeometry>\n");

                Scenario wxScenario = weatherScenarios.get(j);

                ConcurrentHashMap<Integer, ArrayList<Link3D>> increment...
...alPaths =
                        wxScenario.IncrementalPathsByCluster();

                ArrayList<Link3D> path = incrementalPaths.get(i);

                for(Link3D pathLink:path){

                    Node3D<Double> fromNode = pathLink.getFromNode();
                    Node3D<Double> toNode = pathLink.getToNode();

                    s.append("\t\t\t\t<LineString>\n");
                    s.append("\t\t\t\t<altitudeMode>absolute</altitudeM...
...ode>\n");
                    s.append("\t\t\t\t<coordinates>");
                    s.append(fromNode.getLongitude().toString() + ",");...
...                    s.append(fromNode.getLatitude().toString() + ",");
                    s.append(Double.toString(fromNode.getElevation()*10...
...0.0D)+ " ");
                    s.append(toNode.getLongitude().toString() + ",");
                    s.append(toNode.getLatitude().toString() + ",");
                    s.append(Double.toString(toNode.getElevation()*100....
...0D) + "\n");
                    s.append("\t\t\t\t</coordinates>\n");
                    s.append("\t\t\t\t</LineString>\n");
                }
                s.append("\t\t\t\t</MultiGeometry>\n");
```

```
                s.append("\t\t\t</Placemark>\n");
            }
            s.append("\t\t\t</Folder>\n");
        }
        s.append("\t\t</Folder>\n");

        return this;
    }

    /** Write APCDM k-shortest path results to kml file. */
    private FAPioKmlWriter addPathLinksToKml(PrintWriter s, Network n, ...
...Airspace a){
        Scenario flightPlanScenario = n.getFlightPlanScenario();
        ConcurrentHashMap<Integer, ArrayList<ArrayList<Link3D>>> kShort...
...estPathsByCluster =
                flightPlanScenario.KShortestPathsByCluster();

        s.append("\t\t<Folder>\n");
        s.append("\t\t\t<name>K Shortest Path Results</name>\n");
        s.append("\t\t\t<description>K shortest paths by cluster.</desc...
...ription>\n");

        int numClusters = kShortestPathsByCluster.size();

        // Clusters are 1-based
        for(int i=1; i<=numClusters; i++){
            s.append("\t\t\t<Folder>\n");
            s.append("\t\t\t<name>Cluster " + Integer.toString(i) + "</...
...name>\n");

            ArrayList<ArrayList<Link3D>> paths = kShortestPathsByCluste...
...r.get(i);

            for(int j=0; j<paths.size(); j++){
                s.append("\t\t\t<Placemark>\n");
                s.append("\t\t\t<name>Path " + Integer.toString(j) + "<...
.../name>\n");
                s.append("\t\t\t<visibility>0</visibility>\n");
                s.append("\t\t\t\t<styleUrl>#link</styleUrl>\n");
                s.append("\t\t\t\t<Region>\n");
                s.append("\t\t\t\t  <Lod>\n");
                s.append("\t\t\t\t    <minLodPixels>" + MINLODPIXELS + ...
..."</minLodPixels>\n");
                s.append("\t\t\t\t    <maxLodPixels>" + MAXLODPIXELS + ...
..."</maxLodPixels>\n");
                s.append("\t\t\t\t    <minFadeExtent>" + MINFADEEXTENT ...
...+ "</minFadeExtent>\n");
```

```
                s.append("\t\t\t\t    <maxFadeExtent>" + MAXFADEEXTENT ...
...+ "</maxFadeExtent>\n");
                s.append("\t\t\t\t  </Lod>\n");
                s.append("\t\t\t\t</Region>\n");
                s.append("\t\t\t\t<MultiGeometry>\n");

                ArrayList<Link3D> path = paths.get(j);

                for(Link3D pathLink:path){

                    Node3D<Double> fromNode = pathLink.getFromNode();
                    Node3D<Double> toNode = pathLink.getToNode();

                    s.append("\t\t\t\t<LineString>\n");
                    s.append("\t\t\t\t<altitudeMode>absolute</altitudeM...
...ode>\n");
                    s.append("\t\t\t\t<coordinates>");
                    s.append(fromNode.getLongitude().toString() + ",");...
...                    s.append(fromNode.getLatitude().toString() + ",");
                    s.append(Double.toString(fromNode.getElevation()*10...
...0.0D)+ " ");
                    s.append(toNode.getLongitude().toString() + ",");
                    s.append(toNode.getLatitude().toString() + ",");
                    s.append(Double.toString(toNode.getElevation()*100....
...0D) + "\n");
                    s.append("\t\t\t\t</coordinates>\n");
                    s.append("\t\t\t\t</LineString>\n");
                }
                s.append("\t\t\t\t</MultiGeometry>\n");
                s.append("\t\t\t</Placemark>\n");
            }
            s.append("\t\t\t</Folder>\n");
        }
        s.append("\t\t</Folder>\n");

        return this;
    }

    private FAPioKmlWriter addLinksToKml(PrintWriter s, Network n, Airs...
...pace a){
        // Iterate through the link types
        for(Link3D.LinkTypes linkType : Link3D.LinkTypes.values()){
            addLinksToKmlByLinkType(s,n,a,linkType);
        }


        return this;
```

```
    }

    private FAPioKmlWriter addLinksToKmlByLinkType(PrintWriter s, Netwo...
...rk n, Airspace a,
            Link3D.LinkTypes linkType){
        s.append("\t\t<Folder>\n");
        s.append("\t\t\t<name>Network Links (" + linkType.toString() + ...
...")</name>\n");
        s.append("\t\t\t<description>Idealization of airways as links.<...
.../description>\n");


        for(String centerS:FAPio.writeCenters){
            ACenter center = a.getCenter(centerS);
            s.append("\t\t\t<Folder>\n");
            s.append("\t\t\t<name>" + center.getName() + "</name>\n");
            Iterator<ASector> sectorIterator = center.getSectorValueIte...
...rator();
            while(sectorIterator.hasNext()){
                ASector sector = sectorIterator.next();

                addLinksToKmlBySector(s,n,sector,linkType);
            }
            s.append("\t\t\t</Folder>\n");
        }

        if(WRITEEXTERNALAIRWAYS){
            s.append("\t\t\t<Folder>\n");
            s.append("\t\t\t<name>External Network Links</name>\n");
            addLinksToKmlBySector(s,n,ASector.NullSector,linkType);
            s.append("\t\t\t</Folder>\n");
        }


        s.append("\t\t</Folder>\n");

        return this;
    }

    private FAPioKmlWriter addLinksToKmlBySector(PrintWriter s, Network...
... n,
            ASector sector, Link3D.LinkTypes linkType){
        s.append("\t\t\t<Placemark>\n");
        s.append("\t\t\t<name>" + sector.getName() + "</name>\n");
        s.append("\t\t\t<visibility>0</visibility>\n");
        s.append("\t\t\t\t<styleUrl>#link</styleUrl>\n");
        s.append("\t\t\t\t<Region>\n");
```

```
        s.append("\t\t\t\t  <Lod>\n");
        s.append("\t\t\t\t    <minLodPixels>" + MINLODPIXELS + "</minLo...
...dPixels>\n");
        s.append("\t\t\t\t    <maxLodPixels>" + MAXLODPIXELS + "</maxLo...
...dPixels>\n");
        s.append("\t\t\t\t    <minFadeExtent>" + MINFADEEXTENT + "</min...
...FadeExtent>\n");
        s.append("\t\t\t\t    <maxFadeExtent>" + MAXFADEEXTENT + "</max...
...FadeExtent>\n");
        s.append("\t\t\t\t  </Lod>\n");
        s.append("\t\t\t\t</Region>\n");
        s.append("\t\t\t\t<MultiGeometry>\n");


        // Iterate through each link
        Iterator<String> e = n.getLinkIterator();
        //int i=0;
        while(e.hasNext()){
            //if(i++ > 500) break;
            String linkId = e.next();
            Link3D link = n.getLink(linkId);

            // Check for appropriate link type
            if(!link.getLinkType().equals(linkType)){
                continue;
            }

            Node3D<Double> fromNode = link.getFromNode();

            // Only write links for the current sector
            if(!fromNode.getParentSector().equals(sector)){
                continue;
            }

            Node3D<Double> toNode = link.getToNode();

            double fromNodeElevation = fromNode.getElevation()*FLIGHTLE...
...VELTOALTITUDE;
            double toNodeElevation = toNode.getElevation()*FLIGHTLEVELT...
...OALTITUDE;


            s.append("\t\t\t\t<LineString>\n");
            s.append("\t\t\t\t<altitudeMode>absolute</altitudeMode>\n")...
...;
            s.append("\t\t\t\t<coordinates>");
            s.append(fromNode.getLongitude().toString() + ",");
```

```
            s.append(fromNode.getLatitude().toString() + ",");
            s.append(Double.toString(fromNodeElevation) + " ");
            s.append(toNode.getLongitude().toString() + ",");
            s.append(toNode.getLatitude().toString() + ",");
            s.append(Double.toString(toNodeElevation) + "\n");
            s.append("\t\t\t\t</coordinates>\n");
            s.append("\t\t\t\t</LineString>\n");
        }

        s.append("\t\t\t\t</MultiGeometry>\n");
        s.append("\t\t\t</Placemark>\n");


        return this;

    }

    private FAPioKmlWriter addSectorsToKml(PrintWriter s, Network n, Ai...
...rspace a){
        s.append("\t\t<Folder>\n");
        s.append("\t\t\t<name>Sectors</name>\n");
        s.append("\t\t\t<description>East coast sectors.</description>\...
...n");

        // Iterate through each center under consideration
        for(String centerS:FAPio.writeCenters){
            ACenter center = a.getCenter(centerS);
            s.append("\t\t<Folder>\n");
            s.append("\t\t\t<name>" + center.getName() + "</name>\n");
            Iterator<ASector> sectorIterator = center.getSectorValueIte...
...rator();
            while(sectorIterator.hasNext()){
                ASector sector = sectorIterator.next();

                boolean appendEndPlacemark = false;

                // First check to see if at least one module is within ...
...range
                for(int j=0; j < sector.noModules(); j++){
                    ASectorModule module = sector.getModule(j);
                    if(module.getFloor() >= NetworkBuilder.MINSECTORFLO...
...ORALT &&
                            module.getCeiling() >= NetworkBuilder.MINSE...
...CTORCEILALT){
                        s.append("\t\t\t<Placemark>\n");
                        s.append("\t\t\t\t<name>" + sector.getName() + ...
..."</name>\n");
```

```
                        s.append("\t\t\t\t<visibility>0</visibility>\n"...
...);
                        s.append("\t\t\t\t<styleUrl>#sector</styleUrl>\...
...n");
                        s.append("\t\t\t\t<MultiGeometry>\n");
                        appendEndPlacemark = true;
                        break;
                    }
                }

                if(appendEndPlacemark){
                    for(int j=0; j < sector.noModules(); j++){
                        ASectorModule module = sector.getModule(j);
                        if(module.getFloor() < NetworkBuilder.MINSECTOR...
...FLOORALT ||
                                module.getCeiling() < NetworkBuilder.MI...
...NSECTORCEILALT){
                            continue;
                        }

                        // First add the floor and ceiling
                        ArrayList<GPoint3D<Double>> ptsFloor =
                                new ArrayList<GPoint3D<Double>>(module....
...noNodes());
                        ArrayList<GPoint3D<Double>> ptsCeiling =
                                new ArrayList<GPoint3D<Double>>(module....
...noNodes());
                        for(int i=0; i<module.noNodes(); i++){
                            GPoint2D<Double> pt = module.getNode(i);
                            GPoint3D<Double> pt2 = new GPoint3D<Double>...
...(pt.getLongitude(),
                                    pt.getLatitude(), (double)module.ge...
...tFloor());
                            ptsFloor.add(pt2);
                            GPoint3D<Double> pt3 = new GPoint3D<Double>...
...(pt.getLongitude(),
                                    pt.getLatitude(), (double)module.ge...
...tCeiling());
                            ptsCeiling.add(pt3);
                        }
                        this.addPolygon(s, ptsFloor);
                        this.addPolygon(s, ptsCeiling);

                        // Then add the sides
                        for(int i=1; i<module.noNodes(); i++){
                            ArrayList<GPoint3D<Double>> ptsWall =
                                    new ArrayList<GPoint3D<Double>>();
```

```
                            // Two dimensional nodes
                            GPoint2D<Double> pt = module.getNode(i-1);
                            GPoint2D<Double> pt2 = module.getNode(i);

                            // Three dimensional nodes
                            GPoint3D<Double> pta = new GPoint3D<Double>...
...(pt.getLongitude(),
                                    pt.getLatitude(), (double)module.ge...
...tFloor());
                            GPoint3D<Double> ptb = new GPoint3D<Double>...
...(pt2.getLongitude(),
                                    pt2.getLatitude(), (double)module.g...
...etFloor());
                            GPoint3D<Double> ptc = new GPoint3D<Double>...
...(pt2.getLongitude(),
                                    pt2.getLatitude(), (double)module.g...
...etCeiling());
                            GPoint3D<Double> ptd = new GPoint3D<Double>...
...(pt.getLongitude(),
                                    pt.getLatitude(), (double)module.ge...
...tCeiling());

                            ptsWall.add(pta);ptsWall.add(ptb);ptsWall.a...
...dd(ptc);
                            ptsWall.add(ptd);ptsWall.add(pta);

                            this.addPolygon(s, ptsWall);
                        }
                    }

                    s.append("\t\t\t\t</MultiGeometry>\n");
                    s.append("\t\t\t</Placemark>\n");
                }

            }
            s.append("\t\t</Folder>\n");

        }

        s.append("\t\t</Folder>\n");

        return this;
    }

    private FAPioKmlWriter addPolygon(PrintWriter s, ArrayList<GPoint3D...
...<Double>> pts){
        s.append("\t\t\t\t<Polygon>\n");
```

```
        s.append("\t\t\t\t<altitudeMode>absolute</altitudeMode>\n");

        s.append("\t\t\t\t<outerBoundaryIs>\n");
        s.append("\t\t\t\t<LinearRing>\n");
        s.append("\t\t\t\t<coordinates>\n");
        for(GPoint3D<Double> pt:pts){
            double altitude = pt.getElevation() * FLIGHTLEVELTOALTITUDE...
...;
            s.append("\t\t\t\t" + pt.getLongitude().toString() + "," +
                    pt.getLatitude().toString() + "," +
                    Double.toString(altitude) + "\n");
        }
        s.append("\t\t\t\t</coordinates>\n");
        s.append("\t\t\t\t</LinearRing>\n");
        s.append("\t\t\t\t</outerBoundaryIs>\n");

        s.append("\t\t\t\t</Polygon>\n");
        return this;
    }


}
```

FAPioLogWriter.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author Jeff
 */
import java.util.*;
import java.util.concurrent.*;
import java.io.*;
public class FAPioLogWriter {

    public FAPioLogWriter() {
        Network n = APData.network;
        try{
            Scenario flightPlanScenario = n.getFlightPlanScenario();
            ConcurrentHashMap<Integer, ArrayList<ArrayList<Link3D>>> kS...
...hortestPathsByCluster =
                flightPlanScenario.KShortestPathsByCluster();

            int numClusters = kShortestPathsByCluster.size();

            PrintWriter out = new PrintWriter(new BufferedWriter(new Fi...
...leWriter(Settings.logFile)));
            for(int i=1; i<=numClusters; i++){
                out.println("Cluster: " + Integer.toString(i));
                out.println("Planned Links");
                ArrayList<String> flightsInCluster = flightPlanScenario...
....getFlightsInCluster(i);

                for(int j=0; j<flightsInCluster.size(); j++){
                    String flightId = flightsInCluster.get(j);
                    out.println("Flight: " + flightId);

                    for(int k=0; k<flightPlanScenario.noFlightLinks(fli...
...ghtId); k++){

                        Link3D pathLink = flightPlanScenario.getFlightL...
...ink(flightId, k);
                        out.println(pathLink.getIdentifier() + "," +
                                Double.toString(pathLink.getLength()) +...
... "," +
                                Double.toString(pathLink.getDoubleValue...
...(0)));
```

```
                    }


                }

                out.println("");
                out.println("k-Shortest Links");

                ArrayList<ArrayList<Link3D>> paths = kShortestPathsByCl...
...uster.get(i);

                for(int j=0; j<paths.size(); j++){
                    ArrayList<Link3D> path = paths.get(j);

                    for(Link3D pathLink:path){
                        out.println(pathLink.getIdentifier() + "," +
                                Double.toString(pathLink.getLength()) +...
... "," +
                                Double.toString(pathLink.getDoubleValue...
...(0)));
                    }
                }
            }

            out.close();

        }catch(IOException e){
            System.out.println("Could not open log file");
        }
    }
}
```

FAPioNetworkReader.java

```
/*
 * FAPioNetworkReader.java
 *
 * Created on July 29, 2007, 4:53 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author jeff
 */
import java.util.*;
import java.io.*;
public class FAPioNetworkReader {
    Network network_;

    /** Creates a new instance of FAPioNetworkReader */
    public FAPioNetworkReader(Network network) {
        network_ = network;
    }


    public FAPioNetworkReader readLinkLengths() throws IOException{
        BufferedReader d = new BufferedReader(new FileReader(Settings.l...
...inkLengthFile));

        String line;
        while((line = d.readLine()) != null){
            int firstComma = line.indexOf(",");
            int secondComma = line.indexOf(",", firstComma + 1);

            String linkId = line.substring(0, firstComma);
            double linkLength = Double.parseDouble(line.substring(first...
...Comma+1, secondComma));
            double linkAzimuth = Double.parseDouble(line.substring(seco...
...ndComma+1, line.length()));

//            String[] dataFields = line.split(APData.COMMA);
//            String linkId = dataFields[0];
//            double linkLength = Double.parseDouble(dataFields[1]);
//            double linkAzimuth = Double.parseDouble(dataFields[2]);

            if(!network_.hasLink(linkId)) continue;
            Link3D link = network_.getLink(linkId);
```

```
            link.setLength(linkLength);
            link.setAzimuth(linkAzimuth);

            line = null;
        }
        d.close();

        return this;
    }


}
```

FAPioRamsWriter.java

```
/*
 * FAPioRamsWriter.java
 *
 * Created on May 30, 2007, 11:52 AM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author jeff
 */
import java.util.*;
import java.util.concurrent.*;
import java.io.*;
import java.text.*;
public class FAPioRamsWriter {
    private final Airspace a;

    /** Creates a new instance of FAPioRamsWriter */
    public FAPioRamsWriter(Airspace airspace) {
        a = airspace;
    }

    /** Creates input files for RAMS */
    public FAPioRamsWriter writeRamsInputFiles(String directory) throws...
... IOException{
        // Get current date and time
        Calendar cal = Calendar.getInstance(TimeZone.getDefault());
        String DATE_FORMAT = "yyyy-MM-dd 'at' HH:mm:ss";
        java.text.SimpleDateFormat sdf =
                new java.text.SimpleDateFormat(DATE_FORMAT);
        sdf.setTimeZone(TimeZone.getDefault());

        String currentTime = sdf.format(cal.getTime());

        this.writeSectorDat(directory, currentTime);
        this.writeControllerDat(directory, currentTime);
        this.writeCentreSectorDat(directory, currentTime);
        this.writeCentreScheduleDat(directory, currentTime);
        this.writeBoundaryDat(directory, currentTime);
        this.writeCornerDat(directory, currentTime);
        this.writeAirportDat(directory, currentTime);
        this.writeTrafficAndTrafficProfile();
        //this.writeExternalTraffic4D();
```

```
        return this;
    }

    /** Creates airport.dat input file for RAMS */
    public FAPioRamsWriter writeAirportDat(String directory, String cur...
...rentTime) throws IOException{
        String airportDatFilename = directory + File.separator + "airpo...
...rt.dat";
        PrintWriter out = new PrintWriter(new BufferedWriter(new FileWr...
...iter(airportDatFilename)));

        StringBuilder s = new StringBuilder("{ Data File AIRPORT .");
        s.append(File.separator);
        s.append("airport.dat");
        s.append(currentTime);
        s.append(" }\n\n{ Name  Description 3LetterCode Latitude Longit...
...ude Elevation }\n\n");

        Iterator<String> airports = a.getAirportKeyIterator();
        while(airports.hasNext()){
            String airportS = airports.next();
            Airport airport = a.getAirport(airportS);

            s.append(airportS);
            s.append(" \"");
            s.append(airport.getDescription().replace('/',' '));
            s.append("\" ");
            s.append(airportS);
            s.append(" ");
            s.append(airport.getLatitude());
            s.append(" ");
            s.append(airport.getLongitude());
            s.append(" ");
            s.append(airport.getElevation());
            s.append("\n");
        }

        out.print(s.toString());
        out.close();

        s = null;

        return this;
    }

    /** Creates sector.dat input file for RAMS */
```

```
    public FAPioRamsWriter writeSectorDat(String directory, String curr...
...entTime) throws IOException{
        String sectorDatFilename = directory + File.separator + "sector...
....dat";
        PrintWriter out = new PrintWriter(new BufferedWriter(new FileWr...
...iter(sectorDatFilename)));

        /** Stringbuilder object to hold file contents */
        StringBuilder s = new StringBuilder("{ Data File SECTOR .");
        s.append(File.separator);
        s.append("sector.dat ");
        s.append(currentTime);
        s.append(" }\n\n{ Sector ControllerPlanning ControllerTactical ...
...DistanceExtendedAltitude DistanceExtendedBorder TacticalRadarEntryWhen...
...NoSectorPierce }\n\n");

        for(String sc:FAPio.writeCenters){
            ACenter c = a.getCenter(sc);
            Iterator<ASector> eSec = c.getSectorValueIterator();
            while(eSec.hasNext()){
                ASector sec = eSec.next();
                if(sec.getCeiling() >= FAPio.minAltitude){
                    s.append(sec.getName());
                    s.append(" Plan");
                    s.append(sec.getName());
                    s.append(" Tact");
                    s.append(sec.getName());
                    s.append(APData.SPACE);
                    s.append(ASector.DISTANCE_EXTENDED_ALTITUDE);
                    s.append(APData.SPACE);
                    s.append(ASector.DISTANCE_EXTENDED_BORDER);
                    s.append(APData.SPACE);
                    s.append(ASector.TACTICAL_RADAR_ENTRY_WHEN_NO_SECTO...
...R_PIERCE);
                    s.append(APData.SLASH_N);
                }
            }
        }
        out.print(s.toString());
        out.close();

        s = null;

        return this;
    }
```

```
    /** Creates controller.dat input file for RAMS */
    public FAPioRamsWriter writeControllerDat(String directory, String ...
...currentTime) throws IOException{
        String sectorDatFilename = directory + File.separator + "contro...
...ller.dat";
        PrintWriter out = new PrintWriter(new BufferedWriter(new FileWr...
...iter(sectorDatFilename)));

        /** Stringbuilder object to hold file contents */
        StringBuilder s = new StringBuilder("{ Data File CONTROLLER .")...
...;
        s.append(File.separator);
        s.append("controller.dat ");
        s.append(currentTime);
        s.append(" }\n\n{ Controller DistributionEntry DistributionExit...
... VerticalSep DetectionDynamics LateralSeparationDistance LateralSepara...
...tionTime LongitudeSeparationDistance LongitudeSeparationTime RuleSyste...
...m }\n\n");

        for(String sc:FAPio.writeCenters){
            ACenter c = a.getCenter(sc);
            Iterator<ASector> eSec = c.getSectorValueIterator();
            while(eSec.hasNext()){
                ASector sec = eSec.next();
                if(sec.getCeiling() >= FAPio.minAltitude){
                    s.append("Plan");
                    s.append(sec.getName());
                    s.append(" RAMSDefaultDist RAMSDefaultDist RVSMSepa...
...ration DefaultMultipliers ");
                    s.append(ASector.LATERAL_SEPARATION_DISTANCE);
                    s.append(APData.SPACE);
                    s.append(ASector.LATERAL_SEPARATION_TIME);
                    s.append(APData.SPACE);
                    s.append(ASector.LONGITUDE_SEPARATION_DISTANCE);
                    s.append(APData.SPACE);
                    s.append(ASector.LONGITUDE_SEPARATION_TIME);
                    s.append(" PlanningRules");
                    s.append(APData.SLASH_N);

                    s.append("Tact");
                    s.append(sec.getName());
                    s.append(" RAMSDefaultDist RAMSDefaultDist RVSMSepa...
...ration DefaultMultipliers ");
                    s.append(ASector.LATERAL_SEPARATION_DISTANCE);
                    s.append(APData.SPACE);
                    s.append(ASector.LATERAL_SEPARATION_TIME);
                    s.append(APData.SPACE);
```

```
                    s.append(ASector.LONGITUDE_SEPARATION_DISTANCE);
                    s.append(APData.SPACE);
                    s.append(ASector.LONGITUDE_SEPARATION_TIME);
                    s.append(" TacticalRules");
                    s.append(APData.SLASH_N);
                }
            }
        }
        out.print(s.toString());
        out.close();

        s = null;

        return this;
    }


    /** Creates centresector.dat input file for RAMS */
    public FAPioRamsWriter writeCentreSectorDat(String directory, Strin...
...g currentTime) throws IOException{
        String centreSectorDatFilename = directory + File.separator + "...
...centresector.dat";
        PrintWriter out = new PrintWriter(new BufferedWriter(new FileWr...
...iter(centreSectorDatFilename)));

        /** Stringbuilder object to hold file contents */
        StringBuilder s = new StringBuilder("{ Data File CENTRESECTOR ....
...");
        s.append(File.separator);
        s.append("centresector.dat ");
        s.append(currentTime);
        s.append(" }\n\n{ Centre CentreSchedule Sector Boundary Altitud...
...eLow Altitudehigh }\n\n");

        for(String sc:FAPio.writeCenters){
            ACenter c = a.getCenter(sc);
            Iterator<ASector> eSec = c.getSectorValueIterator();
            while(eSec.hasNext()){
                ASector sec = eSec.next();
                if(sec.getCeiling() >= FAPio.minAltitude){
                    for(int j = 0; j < sec.noModules(); j++){
                        ASectorModule secMod = sec.getModule(j);
                        if(secMod.getCeiling() >= FAPio.minAltitude){
                            s.append(sc);
                            s.append(" DefaultSchedule ");
                            s.append(sec.getName());
                            s.append(APData.SPACE);
```

```
                            s.append(secMod.getName());
                            s.append(APData.SPACE);
                            s.append(secMod.getFloor());
                            s.append(APData.SPACE);
                            s.append(secMod.getCeiling());
                            s.append(APData.SLASH_N);
                        }
                    }

                }
            }
        }
        out.print(s.toString());
        out.close();

        s = null;

        return this;
    }

    /** Creates centreschedule.dat input file for RAMS */
    public FAPioRamsWriter writeCentreScheduleDat(String directory, Str...
...ing currentTime) throws IOException{
        String centreScheduleDatFilename = directory + File.separator +...
... "centreschedule.dat";
        PrintWriter out = new PrintWriter(new BufferedWriter(new FileWr...
...iter(centreScheduleDatFilename)));

        /** Stringbuilder object to hold file contents */
        StringBuilder s = new StringBuilder("{ Data File CENTRESCHEDULE...
... .");
        s.append(File.separator);
        s.append("centreschedule.dat ");
        s.append(currentTime);
        s.append(" }\n\n{ Centre CentreSchedule DayOfWeek Configuration...
...StartTime ConfigurationEndTime }\n\n");

        for(String sc:FAPio.writeCenters){
            s.append(sc);
            s.append(" DefaultSchedule SunToSat 000000 240000\n");
        }
        out.print(s.toString());
        out.close();

        s = null;

        return this;
```

```
    }
//    /** Creates centresector.dat input file for RAMS */
//    public FAPioRamsWriter writeCentreSectorDat(String directory, Str...
...ing currentTime) throws IOException{
//        String centreSectorDatFilename = directory + File.separator +...
... "centresector.dat";
//        PrintWriter out = new PrintWriter(new BufferedWriter(new File...
...Writer(centreSectorDatFilename)));
//
//        /** Stringbuilder object to hold file contents */
//        StringBuilder s = new StringBuilder("{ Data File CENTRESECTOR...
... .");
//        s.append(File.separator);
//        s.append("centresector.dat ");
//        s.append(currentTime);
//        s.append(" }\n\n{ Centre CentreSchedule Sector Boundary Altit...
...udeLow Altitudehigh }\n\n");
//
//        for(String sc:FAPio.writeCenters){
//            ACenter c = a.getCenter(sc);
//            Iterator<ASector> eSec = c.getSectorValueIterator();
//            while(eSec.hasNext()){
//                ASector sec = eSec.next();
//                if(sec.getCeiling() >= FAPio.minAltitude){
//                    for(int j = 0; j < sec.noModules(); j++){
//                        ASectorModule secMod = sec.getModule(j);
//                        if(secMod.getCeiling() >= FAPio.minAltitude){...
...//
//                            for(int hour = 0; hour <= 23; hour++){
//                                for(int minute = 0; minute <= 45; min...
...ute = minute+15){
//                                    int nextHour = hour;
//                                    int nextMinute = minute + 15;
//                                    if(minute == 45){
//                                        nextHour = hour+1;
//                                        nextMinute = 0;
//                                    }
//
//                                    s.append(sec.getName());
//                                    s.append(" Schedule");
//                                    if(hour < 10) s.append("0");
//                                    s.append(hour);
//                                    s.append(minute);
//                                    s.append("To");
//                                    if(nextHour < 10) s.append("0");
//                                    s.append(nextHour);
//                                    s.append(nextMinute);
```

```
//                                    s.append(" ");
//                                    s.append(sec.getName());
//                                    s.append(APData.SPACE);
//                                    s.append(secMod.getName());
//                                    s.append(APData.SPACE);
//                                    s.append(secMod.getFloor());
//                                    s.append(APData.SPACE);
//                                    s.append(secMod.getCeiling());
//                                    s.append(APData.SLASH_N);
//                                }
//                            }
//                        }
//                    }
//
//                }
//            }
//        }
//        out.print(s.toString());
//        out.close();
//
//        s = null;
//
//        return this;
//    }
//
//    /** Creates centreschedule.dat input file for RAMS */
//    public FAPioRamsWriter writeCentreScheduleDat(String directory, S...
...tring currentTime) throws IOException{
//        String centreScheduleDatFilename = directory + File.separator...
... + "centreschedule.dat";
//        PrintWriter out = new PrintWriter(new BufferedWriter(new File...
...Writer(centreScheduleDatFilename)));
//
//        /** Stringbuilder object to hold file contents */
//        StringBuilder s = new StringBuilder("{ Data File CENTRESCHEDU...
...LE .");
//        s.append(File.separator);
//        s.append("centreschedule.dat ");
//        s.append(currentTime);
//        s.append(" }\n\n{ Centre CentreSchedule DayOfWeek Configurati...
...onStartTime ConfigurationEndTime }\n\n");
//
//        for(String sc:FAPio.writeCenters){
//            ACenter c = a.getCenter(sc);
//            Iterator<ASector> eSec = c.getSectorValueIterator();
//            while(eSec.hasNext()){
//                ASector sec = eSec.next();
```

```
//
//                for(int hour = 0; hour <= 23; hour++){
//                    for(int minute = 0; minute <= 45; minute = minute...
... + 15){
//                        int nextHour = hour;
//                        int nextMinute = minute + 15;
//                        if(minute == 45){
//                            nextHour = hour + 1;
//                            nextMinute = 0;
//                        }
//
//                        s.append(sec.getName());
//                        s.append(" Schedule");
//                        if (hour < 10) {
//                            s.append("0");
//                        }
//                        s.append(hour);
//                        if(minute == 0) s.append("0");
//                        s.append(minute);
//                        s.append("To");
//                        if (nextHour < 10) {
//                            s.append("0");
//                        }
//                        s.append(nextHour);
//                        if(nextMinute == 0) s.append("0");
//                        s.append(nextMinute);
//                        s.append(" SunToSat ");
//
//                        if(hour < 10) s.append("0");
//                        s.append(hour);
//                        if(minute == 0) s.append("0");
//                        s.append(minute);
//                        s.append("00 ");
//
//                        if(nextHour < 10) s.append("0");
//                        s.append(nextHour);
//                        if(nextMinute == 0) s.append("0");
//                        s.append(nextMinute);
//                        s.append("00\n");
//                    }
//                }
//            }
//        }
//        out.print(s.toString());
//        out.close();
//
//        s = null;
```

```
//
//        return this;
//    }


    /** Write sector capacity values for simulation */
    public FAPioRamsWriter writeThresholdCapacity(){
        String thresholdFile = Settings.ramsFilesDir + File.separator +...
... "threshold.dat";

        Scenario wxScenario = APData.network.getWeatherScenario(0);

        // Time period for capacity (seconds)
        final int timePeriod = 60 * 15;

        try{
            PrintWriter outThreshold = new PrintWriter(new BufferedWrit...
...er(new FileWriter(thresholdFile)));

            for (String sc : FAPio.writeCenters) {
                ACenter c = a.getCenter(sc);
                Iterator<ASector> eSec = c.getSectorValueIterator();
                while (eSec.hasNext()) {
                    ASector sec = eSec.next();

                    outThreshold.print(sec.getName());
                    int mapValue = sec.getMapValue();
                    String mapS = Integer.toString(mapValue);
                    // Long Term Minimum
                    // Long Term Maximum
                    // Long Term Time Period
                    // Short Term Minimum
                    // Short Term Maximum
                    // Short Term Time Period
                    if(mapValue <= 5){
                        outThreshold.println(" 0 " + mapS + " 180 0 " +...
... mapS + " 180");
                    }else if(mapValue <= 7){
                        outThreshold.println(" 0 " + mapS + " 240 0 " +...
... mapS + " 240");
                    }else if(mapValue <= 8){
                        outThreshold.println(" 0 " + mapS + " 300 0 " +...
... mapS + " 300");
                    }else if(mapValue <= 10){
                        outThreshold.println(" 0 " + mapS + " 360 0 " +...
... mapS + " 360");
                    }else if(mapValue <= 12){
```

```
                    outThreshold.println(" 0 " + mapS + " 420 0 " +...
... mapS + " 420");
                }else if(mapValue <= 13){
                    outThreshold.println(" 0 " + mapS + " 480 0 " +...
... mapS + " 480");
                }else if(mapValue <= 15){
                    outThreshold.println(" 0 " + mapS + " 540 0 " +...
... mapS + " 540");
                }else if(mapValue <= 17){
                    outThreshold.println(" 0 " + mapS + " 600 0 " +...
... mapS + " 600");
                }else{ // if(mapValue <= 18)}
                    outThreshold.println(" 0 " + mapS + " 660 0 " +...
... mapS + " 660");
                }

//                  int timeInterval = 0;
//                  int sectorIndex = sec.getArrayIndex();
//
//                  for (int hour = 0; hour <= 23; hour++) {
//                      for (int minute = 0; minute <= 45; minute = m...
...inute + 15) {
//                          int nextHour = hour;
//                          int nextMinute = minute + 15;
//                          if (minute == 45) {
//                              nextHour = hour + 1;
//                              nextMinute = 0;
//                          }
//
//                          outThreshold.print("Schedule");
//                          if (hour < 10) {
//                              outThreshold.print("0");
//                          }
//                          outThreshold.print(hour);
//                          if (minute == 0) {
//                              outThreshold.print("0");
//                          }
//                          outThreshold.print(minute);
//                          outThreshold.print("To");
//                          if (nextHour < 10) {
//                              outThreshold.print("0");
//                          }
//                          outThreshold.print(nextHour);
//                          if (nextMinute == 0) {
//                              outThreshold.print("0");
//                          }
//                          outThreshold.print(nextMinute);
```

```
//
//                            // Use a long term minimum threshold of 0...
...//                            outThreshold.print(" 0 ");
//
//                            // Get sector capacity
//                            double sectorCapacity = wxScenario.getSec...
...torCapacity15Min(sectorIndex, timeInterval);
//                            long sectorCapacityL = Math.round(Math.ce...
...il(sectorCapacity));
//                            outThreshold.print(sectorCapacityL);
//
//                            //
//                            outThreshold.print(" ");
//
//                            // Print the time duration
//                            outThreshold.print(timePeriod);
//
//                            // Use same parameters for short term thr...
...esholds
//                            outThreshold.print(" 0 ");
//                            outThreshold.print(sectorCapacityL);
//                            outThreshold.print(" ");
//                            outThreshold.println(timePeriod);
//
//                            // Increment the time interval
//                            timeInterval++;
//                        }
//                    }
                }
            }

            outThreshold.close();
        }catch(IOException e){
            System.out.println("Could not write RAMS threshold file!");...
...        }

        return this;
    }


    /** Creates boundary.dat input file for RAMS */
    public FAPioRamsWriter writeBoundaryDat(String directory, String cu...
...rrentTime) throws IOException{
        String boundaryDatFilename = directory + File.separator + "boun...
...dary.dat";
        PrintWriter out = new PrintWriter(new BufferedWriter(new FileWr...
...iter(boundaryDatFilename)));
```

```
        /** Stringbuilder object to hold file contents */
        StringBuilder s = new StringBuilder("{ Data File BOUNDARY .");
        s.append(File.separator);
        s.append("boundary.dat ");
        s.append(currentTime);
        s.append(" }\n\n{ Boundary Corner_i Corner_j }\n\n");

        for(String sc:FAPio.writeCenters){
            ACenter c = a.getCenter(sc);
            Iterator<ASector> eSec = c.getSectorValueIterator();
            while(eSec.hasNext()){
                ASector sec = eSec.next();
                if(sec.getCeiling() >= FAPio.minAltitude){
                    for(int i = 0; i < sec.noModules(); i++){
                        ASectorModule secMod = sec.getModule(i);
                        if(secMod.getCeiling() < FAPio.minAltitude){con...
...tinue;}

                        /* Reason for secMod.noNodes()-1
                         * Don't write the last point it just
                         * duplicates the first point. Use the
                         * first point for closure.
                         */
                        for(int j = 0; j < secMod.noNodes()-1; j++){
                            int j_plus_1 = j+1 < secMod.noNodes()-1? j+...
...1:0;
                            s.append(secMod.getName());
                            s.append(APData.SPACE);
                            s.append(secMod.getName());
                            s.append(APData.UNDERSCORE);
                            s.append(j);
                            s.append(APData.SPACE);
                            s.append(secMod.getName());
                            s.append(APData.UNDERSCORE);
                            s.append(j_plus_1);
                            s.append(APData.SLASH_N);
                        }
                    }
                }
            }

        }
        out.print(s.toString());
        out.close();

        s = null;
```

```
        return this;
    }

    /** Creates corner.dat input file for RAMS */
    public FAPioRamsWriter writeCornerDat(String directory, String curr...
...entTime) throws IOException{
        String cornerDatFilename = directory + File.separator + "corner...
....dat";
        PrintWriter out = new PrintWriter(new BufferedWriter(new FileWr...
...iter(cornerDatFilename)));

        /** Stringbuilder object to hold file contents */
        StringBuilder s = new StringBuilder("{ Data File CORNER .");
        s.append(File.separator);
        s.append("corner.dat ");
        s.append(currentTime);
        s.append(" }\n\n{ Corner Latitude Longitude }\n\n");

        for(String sc:FAPio.writeCenters){
            ACenter c = a.getCenter(sc);
            Iterator<ASector> eSec = c.getSectorValueIterator();
            while(eSec.hasNext()){
                ASector sec = eSec.next();
                if(sec.getCeiling() >= FAPio.minAltitude){
                    for(int i = 0; i < sec.noModules(); i++){
                        ASectorModule secMod = sec.getModule(i);
                        if(secMod.getCeiling() < FAPio.minAltitude){con...
...tinue;}
                        for(int j = 0; j < secMod.noNodes(); j++){
                            s.append(secMod.getName());
                            s.append(APData.UNDERSCORE);
                            s.append(j);
                            s.append(APData.SPACE);

                            GPoint2D p = secMod.getNode(j);
                            s.append(p.getLatitude());
                            s.append(APData.SPACE);
                            s.append(p.getLongitude());
                            s.append(APData.SLASH_N);
                        }
                    }
                }
            }
        }
        out.print(s.toString());
        out.close();
```

```
        s = null;

        return this;
    }

    /** Write the flight plan trafficprofile for a flight */
    public void writeFlightPlanProfile(Flight f, PrintWriter out){
        /** Latitude/Longitude decimal format */
        DecimalFormat format = new DecimalFormat("###0.0000000");

        // Add planned traffic profile data
        // Write the origin airport
        out.print(f.getCallSign());
        out.print(" ");
        out.print(format.format(f.getOriginAirport().getLatitude()));
        out.print(" ");
        out.print(format.format(f.getOriginAirport().getLongitude()));
        out.print(" 000000 1\n");
        FTrajectory trajectory = f.getPlannedTrajectory();
        for (int i = 0; i < trajectory.noWaypoints(); i++) {
            GPoint4D<Double> wpt = trajectory.getWaypoint(i);
            out.print(f.getCallSign());
            out.print(" ");
            out.print(format.format(wpt.getLatitude()));
            out.print(" ");
            out.print(format.format(wpt.getLongitude()));
            out.print(" 000000 ");
            out.print(i + 2);
            out.print("\n");
        }
        // Write the destination airport
        out.print(f.getCallSign());
        out.print(" ");
        out.print(format.format(f.getDestinationAirport().getLatitude()...
...));
        out.print(" ");
        out.print(format.format(f.getDestinationAirport().getLongitude(...
...)));
        out.print(" 000000 ");
        out.print(trajectory.noWaypoints() + 2);
        out.print("\n");
    }

    /* Creates the traffic.dat and trafficprofile.dat based on the weat...
...her
     * routing.
```

```
     */
    public FAPioRamsWriter writeWeatherTrafficAndTrafficProfile(int num...
...KPaths){
        Network n = APData.network;
        int numWxScenarios = n.getNumberWeatherScenarios();
        Scenario flightPlanScenario = n.getFlightPlanScenario();
        ConcurrentHashMap<Integer, ArrayList<ArrayList<Link3D>>> kShort...
...estPathsByCluster =
                flightPlanScenario.KShortestPathsByCluster();

        String wxTrafficFilename = Settings.ramsFilesDir + File.separat...
...or + "traffic_weather.dat";
        String wxProfileFilename = Settings.ramsFilesDir + File.separat...
...or + "trafficprofile_weather.dat";

        // Create one filename for each of the k-shortest paths
        ArrayList<String> wxApcdmFilenames = new ArrayList<String>(numK...
...Paths);
        for(int i=0; i<numKPaths; i++){
            String wxApcdmFilename = Settings.ramsFilesDir + File.separ...
...ator +
                    "trafficprofile_apcdm" + Integer.toString(i) + ".da...
...t";
            wxApcdmFilenames.add(wxApcdmFilename);
        }

        // Create one filename for each of the incremental scenarios
        ArrayList<String> wxIncrementalFilenames = new ArrayList<String...
...>(numWxScenarios);
        for(int i=0; i<numWxScenarios; i++){
            String wxIncrementalFilename = Settings.ramsFilesDir + File...
....separator +
                    "trafficprofile_incremental" + Integer.toString(i) ...
...+ ".dat";
            wxIncrementalFilenames.add(wxIncrementalFilename);
        }

        // List of flights that go through the flow constrained sectors...
...        ArrayList<Flight> flightsLt18k = n.getConstrainedFlightsLessTha...
...n18kft();
        ArrayList<Flight> flightsGt18k = n.getConstrainedFlightsGreater...
...18kft();
        ArrayList<Flight> allFlights = new ArrayList<Flight>(flightsLt1...
...8k);
        allFlights.addAll(flightsGt18k);

        /** Latitude/Longitude decimal format */
```

```
        DecimalFormat format = new DecimalFormat("###0.0000000");

        /** Count of flight plan useages over shortest path */
        int countFlightPlanOverPath = 0;

        try {
            PrintWriter outWxTraffic = new PrintWriter(new BufferedWrit...
...er(new FileWriter(wxTrafficFilename)));
            PrintWriter outWxProfile = new PrintWriter(new BufferedWrit...
...er(new FileWriter(wxProfileFilename)));

            ArrayList<PrintWriter> outWxApcdms = new ArrayList<PrintWri...
...ter>(numKPaths);
            for(int i=0; i<numKPaths; i++){
                PrintWriter outWxApcdm = new PrintWriter(new BufferedWr...
...iter(new FileWriter(wxApcdmFilenames.get(i))));
                outWxApcdms.add(outWxApcdm);
            }

            ArrayList<PrintWriter> outWxIncrementals = new ArrayList<Pr...
...intWriter>(numWxScenarios);
            for(int i=0; i<numWxScenarios; i++){
                PrintWriter outWxIncremental = new PrintWriter(new Buff...
...eredWriter(new FileWriter(wxIncrementalFilenames.get(i))));
                outWxIncrementals.add(outWxIncremental);
            }
            for (Flight f : allFlights) {
                // Only consider flights that have a flight plan
                if (!f.hasPlannedTrajectory()) {
                    continue;
                }
                // Add traffic data
                String flightCallsign = f.getCallSign();
                String flightCallsign2 = flightCallsign.split("_")[0];
                String departureTimeString = a.hasFlightDepartureTime(f...
...lightCallsign2) ? a.getFlightDepartureTime(flightCallsign2) : f.getDep...
...artureTimeString();
                outWxTraffic.print(departureTimeString);
                outWxTraffic.print(" ");
                outWxTraffic.print(flightCallsign);
                outWxTraffic.print(" ");
                outWxTraffic.print(f.getOriginAirport().getIdentifier()...
...);
                outWxTraffic.print(" RWY ");
                outWxTraffic.print(f.getDestinationAirport().getIdentif...
...ier());
                outWxTraffic.print(" RWY ");
```

```
                outWxTraffic.print(f.getAircraftType().getCallSign());
                outWxTraffic.print(" C DefaultACNavEquipment 0 ");
                outWxTraffic.print(f.getCruisingFlightLevel());
                outWxTraffic.print(" 0\n");

                // Add planned traffic profile data
                writeFlightPlanProfile(f, outWxProfile);


                // Create k-shortest paths (APCDM) traffic profiles
                for(int i=0; i<numKPaths; i++){
                    PrintWriter outWxApcdm = outWxApcdms.get(i);

                    // Check if should use the flight plan
                    if(flightPlanScenario.useFlightPlan(flightCallsign)...
...){
                        writeFlightPlanProfile(f, outWxApcdm);
                        if(i == 0) countFlightPlanOverPath++;
                        continue;
                    }

                    // Get cluster index for flight
                    int clusterIdx = flightPlanScenario.getClusterIndex...
...ForFlight(flightCallsign);

                    // Get paths
                    ArrayList<ArrayList<Link3D>> paths = kShortestPaths...
...ByCluster.get(clusterIdx);

                    // Get path
                    ArrayList<Link3D> path = paths.get(i);

                    // Write the path
                    for(int j=0; j<path.size(); j++){
                        Link3D link = path.get(j);
                        if(j == 0){
                            Node3D<Double> pt = link.getFromNode();

                            outWxApcdm.print(f.getCallSign());
                            outWxApcdm.print(" ");
                            outWxApcdm.print(format.format(pt.getLatitu...
...de()));
                            outWxApcdm.print(" ");
                            outWxApcdm.print(format.format(pt.getLongit...
...ude()));
                            outWxApcdm.print(" 000000 ");
                            outWxApcdm.print(j + 1);
```

```
                            outWxApcdm.print("\n");
                        }
                        Node3D<Double> pt = link.getToNode();
                        outWxApcdm.print(f.getCallSign());
                        outWxApcdm.print(" ");
                        outWxApcdm.print(format.format(pt.getLatitude()...
...));
                        outWxApcdm.print(" ");
                        outWxApcdm.print(format.format(pt.getLongitude(...
...)));
                        outWxApcdm.print(" 000000 ");
                        outWxApcdm.print(j + 2);
                        outWxApcdm.print("\n");
                    }
                }

                // Create incremental traffic profiles
                for(int i=0; i<numWxScenarios; i++){
                    PrintWriter outWxIncremental = outWxIncrementals.ge...
...t(i);
                    // Check if should use the flight plan
                    if(flightPlanScenario.useFlightPlan(flightCallsign)...
...){
                        writeFlightPlanProfile(f, outWxIncremental);
                        if(i == 0) countFlightPlanOverPath++;
                        continue;
                    }

                    // Get cluster index for flight
                    int clusterIdx = flightPlanScenario.getClusterIndex...
...ForFlight(flightCallsign);

                    // Get incremental path
                    Scenario wxScenario = n.getWeatherScenario(i);
                    ConcurrentHashMap<Integer, ArrayList<Link3D>> incre...
...mentalPathsByCluster =
                            wxScenario.IncrementalPathsByCluster();
                    ArrayList<Link3D> path = incrementalPathsByCluster....
...get(clusterIdx);

                    // Write the path
                    for(int j=0; j<path.size(); j++){
                        Link3D link = path.get(j);
                        if(j == 0){
                            Node3D<Double> pt = link.getFromNode();

                            outWxIncremental.print(f.getCallSign());
```

```
                            outWxIncremental.print(" ");
                            outWxIncremental.print(format.format(pt.get...
...Latitude()));
                            outWxIncremental.print(" ");
                            outWxIncremental.print(format.format(pt.get...
...Longitude()));
                            outWxIncremental.print(" 000000 ");
                            outWxIncremental.print(j + 1);
                            outWxIncremental.print("\n");
                        }
                        Node3D<Double> pt = link.getToNode();
                        outWxIncremental.print(f.getCallSign());
                        outWxIncremental.print(" ");
                        outWxIncremental.print(format.format(pt.getLati...
...tude()));
                        outWxIncremental.print(" ");
                        outWxIncremental.print(format.format(pt.getLong...
...itude()));
                        outWxIncremental.print(" 000000 ");
                        outWxIncremental.print(j + 2);
                        outWxIncremental.print("\n");
                    }
                }
            }


            outWxTraffic.close();
            outWxProfile.close();

            for(int i=0; i<numKPaths; i++){
                outWxApcdms.get(i).close();
            }

            for(int i=0; i<numWxScenarios; i++){
                outWxIncrementals.get(i).close();
            }
        } catch (IOException e) {
            System.out.println("Could not write RAMS weather traffic an...
...d traffic profile!");
        }

        System.out.print("Number of times the flight plan route is sele...
...cted over the shortest route: ");
        System.out.println(countFlightPlanOverPath);

        return this;
    }
```

```
    /* Creates the traffic.dat and trafficprofile.dat for input into RA...
...MS
     */
    public FAPioRamsWriter writeTrafficAndTrafficProfile() throws IOExc...
...eption{
        String trafficFilename = Settings.ramsFilesDir + File.separator...
... + "traffic.dat";
        String trafficProfileFilename = Settings.ramsFilesDir + File.se...
...parator + "trafficprofile.dat";
        PrintWriter outTraffic = new PrintWriter(new BufferedWriter(new...
... FileWriter(trafficFilename)));
        PrintWriter outTrafficProfile = new PrintWriter(new BufferedWri...
...ter(new FileWriter(trafficProfileFilename)));

        /** Stringbuilder object to hold file contents */
        StringBuilder sTraffic = new StringBuilder();
        StringBuilder sTrafficProfile = new StringBuilder();

        Iterator<String> e = a.getFlightIterator();

        /** Latitude/Longitude decimal format */
        DecimalFormat format = new DecimalFormat("###0.00000000");

        while(e.hasNext()){
            Flight f = a.getFlight(e.next());

            // Only consider flights that have a flight plan
            if(!f.hasPlannedTrajectory()) continue;

            // Add traffic data
            sTraffic.append(f.getDepartureTimeString());
            sTraffic.append(" ");
            sTraffic.append(f.getCallSign());
            sTraffic.append(" ");
            sTraffic.append(f.getOriginAirport().getIdentifier());
            sTraffic.append(" RWY ");
            sTraffic.append(f.getDestinationAirport().getIdentifier());...
...             sTraffic.append(" RWY ");
            sTraffic.append(f.getAircraftType().getCallSign());
            sTraffic.append(" C DefaultACNavEquipment 0 ");
            sTraffic.append(f.getCruisingFlightLevel());
            sTraffic.append(" 0\n");

            // Add traffic profile data
            FTrajectory trajectory = f.getPlannedTrajectory();
            for(int i=0; i<trajectory.noWaypoints(); i++){
```

```
                GPoint4D<Double> wpt = trajectory.getWaypoint(i);

                sTrafficProfile.append(f.getCallSign());
                sTrafficProfile.append(" ");
                sTrafficProfile.append(format.format(wpt.getLatitude())...
...);
                sTrafficProfile.append(" ");
                sTrafficProfile.append(format.format(wpt.getLongitude()...
...));
                sTrafficProfile.append(" 000000 ");
                sTrafficProfile.append(i+1);
                sTrafficProfile.append("\n");
            }

        }

        outTraffic.print(sTraffic.toString());
        outTrafficProfile.print(sTrafficProfile.toString());

        outTraffic.close();
        outTrafficProfile.close();
        return this;
    }

    /* Creates the ExternalTraffic4D file for input into RAMS
    */
    public FAPioRamsWriter writeExternalTraffic4D() throws IOException{...
...        String externalTraffic4DFilename = Settings.ramsFilesDir + File...
....separator + "externaltraffic4d.dat";
        PrintWriter outTraffic = new PrintWriter(new BufferedWriter(new...
... FileWriter(externalTraffic4DFilename)));

        /** Stringbuilder object to hold file contents */
        StringBuilder sTraffic = new StringBuilder();

        Iterator<String> e = a.getFlightIterator();

        double previousAltitude = 0.0;
        double currentAltitude = 0.0;
        final double SECONDS_TO_HOURS = 1.0/3600.0D;
        while(e.hasNext()){
            Flight f = a.getFlight(e.next());
            FTrajectory t = f.getRadarTrackTrajectory();

            for (int i=0; i<t.noWaypoints(); i++){
                // 4-D Point on Trajectory
                //
```

```
                GPoint4D<Double> p = t.getWaypoint(i);

                if(p.getElevation() < 0.0D){
                    currentAltitude = previousAltitude;
                } else{
                    currentAltitude = p.getElevation();
                    previousAltitude = currentAltitude;
                }

                GregorianCalendar gc = p.getTime();
                double t_seconds = gc.get(gc.HOUR_OF_DAY)*3600.0 + gc.g...
...et(gc.MINUTE)*60.0 + gc.get(gc.SECOND);

                // Check to see if flights are outside the filter range...
... (minutes)
                double t_minutes = t_seconds*SECONDS_TO_HOURS;

                if(Settings.filterTime){
                    if(t_minutes < Settings.filterMinTime || t_minutes ...
...> Settings.filterMaxTime){
                        continue;
                    }
                }

                /*if(Settings.filterAltitude){
                    if(currentAltitude < Settings.filterMinAltitude || ...
...currentAltitude > Settings.filterMaxAltitude){
                        continue;
                    }
                }*/

                sTraffic.append(f.getCallSign()); sTraffic.append(APDat...
...a.SPACE);
                sTraffic.append(p.getLatitude()); sTraffic.append(APDat...
...a.SPACE);
                sTraffic.append(p.getLongitude()); sTraffic.append(APDa...
...ta.SPACE);
                sTraffic.append(currentAltitude); sTraffic.append(APDat...
...a.SPACE);
                sTraffic.append(t_seconds); sTraffic.append(APData.SPAC...
...E);
                sTraffic.append(i+1);
                sTraffic.append(APData.SLASH_N);
            }
        }
        outTraffic.print(sTraffic.toString());
        outTraffic.close();
```

```
        sTraffic = null;

        return this;
    }



    /*  Creates the Restriction, RestrictBoundary, Boundary, and Corner...
... files
     *  to define the weather restriction.
     */
    public FAPioRamsWriter writeWeatherRestriction(){
        String restrictionFile = Settings.ramsFilesDir + File.separator...
... + "restriction.dat";
        String restrictBoundaryFile = Settings.ramsFilesDir + File.sepa...
...rator + "restrictedboundary.dat";
        String boundaryFile = Settings.ramsFilesDir + File.separator + ...
..."boundary.dat";
        String cornerFile = Settings.ramsFilesDir + File.separator + "c...
...orner.dat";

        // boundary and corner file should be appended
        boolean append = true;

        try{
            PrintWriter outRestriction = new PrintWriter(new BufferedWr...
...iter(new FileWriter(restrictionFile)));
            PrintWriter outRestrictBoundary = new PrintWriter(new Buffe...
...redWriter(new FileWriter(restrictBoundaryFile)));
            PrintWriter outBoundary = new PrintWriter(new BufferedWrite...
...r(new FileWriter(boundaryFile, append)));
            PrintWriter outCorner = new PrintWriter(new BufferedWriter(...
...new FileWriter(cornerFile, append)));

            // Iterate through each of the convection forecasts
            for(int timeOfForecast:APData.convectionForecasts.keySet())...
...{
                // Get the forecast object
                ConvectionForecast convectionForecast = APData.convecti...
...onForecasts.get(timeOfForecast);

                // Get polygons of observed convection with 10 nm buffe...
...r
                ArrayList<GPoint2D<Double>> observedConvection10NmBuffe...
...rPolygon =
```

```
                        convectionForecast.observedConvection10NmBuffer...
...Polygon();

                // Get the observed convection tops
                // 3: 370+
                // 2: 310-370
                // 1: 250-310
                // 0: < 250
                ArrayList<Integer> observedConvectionTops =
                        convectionForecast.observedConvectionTops();

                // Get the locations of the observed convection tops
                ArrayList<GPoint2D<Double>> observedConvectionTopLocati...
...ons =
                        convectionForecast.observedConvectionTopLocatio...
...ns();

                // Forecast is one hour ahead
                int startOfRestriction = timeOfForecast + 1;
                int endOfRestriction = timeOfForecast + 2;

                // Add restriction to restriction.dat
                String restrictionName = "WeatherRestriction" + Integer...
....toString(timeOfForecast);
                outRestriction.print(restrictionName);
                outRestriction.print(" ");
                outRestriction.print(startOfRestriction);
                outRestriction.print("0000 ");
                outRestriction.print(endOfRestriction);
                outRestriction.println("0000 SunToSat");

                // Break the observed convection into groups
                ArrayList<GPoint2D<Double>> convectionGroup =
                        new ArrayList<GPoint2D<Double>>();
                ArrayList<ArrayList<GPoint2D<Double>>> convectionGroups...
... =
                        new ArrayList<ArrayList<GPoint2D<Double>>>();
                ArrayList<Integer> convectionTops = new ArrayList<Integ...
...er>();
                for(int i=0; i<observedConvection10NmBufferPolygon.size...
...(); i++){
                    GPoint2D<Double> pt = observedConvection10NmBufferP...
...olygon.get(i);

                    if(Double.isNaN(pt.getLatitude()) ||
                            i == observedConvection10NmBufferPolygon.si...
...ze() - 1){
```

```
                        if(i == observedConvection10NmBufferPolygon.siz...
...e() - 1){
                            convectionGroup.add(pt);
                        }

                        // Find the closest convection tops for this gr...
...oup
                        GPoint2D<Double> firstPt = convectionGroup.get(...
...0);
                        int closestIdx = 0;
                        double closestDist = Double.MAX_VALUE;

                        for(int j=0; j<observedConvectionTopLocations.s...
...ize(); j++){
                            GPoint2D<Double> observedPt = observedConve...
...ctionTopLocations.get(j);

                            Mapping m = new Mapping(firstPt.getLatitude...
...(),
                                    firstPt.getLongitude(), observedPt....
...getLatitude(),
                                    observedPt.getLongitude());

                            double dist = m.getDistance();

                            if(dist < closestDist){
                                closestDist = dist;
                                closestIdx = j;
                            }
                        }

                        // Get the convection tops for this group
                        int tops = observedConvectionTops.get(closestId...
...x);
                        convectionTops.add(tops);

                        // Add to list of groups
                        convectionGroups.add(convectionGroup);
                        // Reset the convection group list
                        convectionGroup = new ArrayList<GPoint2D<Double...
...>>();
                    }else{
                        convectionGroup.add(pt);
                    }
                }

                // Add boundaries to restrictedboundary.dat
```

```
                for(int i=0; i<convectionTops.size(); i++){
                    outRestrictBoundary.print(restrictionName);
                    outRestrictBoundary.print(" ");

                    String boundaryName = "WR" + Integer.toString(timeO...
...fForecast)
                            + "_B" + Integer.toString(i);

                    outRestrictBoundary.print(boundaryName);
                    outRestrictBoundary.print(" ");

                    // Lower altitude of the convection (1000 ft)
                    outRestrictBoundary.print("10 ");

                    // Get the tops
                    int tops = convectionTops.get(i);
                    if(tops == 0){
                        // Use 12,500 ft
                        outRestrictBoundary.println("125");
                    }else if(tops == 1){
                        // Use 28,000 ft
                        outRestrictBoundary.println("280");
                    }else if(tops == 2){
                        // Use 34,000 ft
                        outRestrictBoundary.println("340");
                    }else{ // tops == 3
                        outRestrictBoundary.println("400");
                    }
                }


                // Add corners to boundary.dat
                for(int i=0; i<convectionGroups.size(); i++){
                    ArrayList<GPoint2D<Double>> convectionGroupi =
                            convectionGroups.get(i);

                    int numCorners = convectionGroupi.size();

                    String boundaryName = "WR" + Integer.toString(timeO...
...fForecast)
                            + "_B" + Integer.toString(i);

                    for(int j=0; j<numCorners; j++){
                        String corner1Name = "WR" + Integer.toString(ti...
...meOfForecast)
                            + "_B" + Integer.toString(i) + "_C" + Integ...
...er.toString(j);
```

```
                        String corner2Name;
                        if(j == numCorners-1){
                            corner2Name = "WR" + Integer.toString(timeO...
...fForecast)
                            + "_B" + Integer.toString(i) + "_C" + Integ...
...er.toString(0);
                        }else{
                            corner2Name = "WR" + Integer.toString(timeO...
...fForecast)
                            + "_B" + Integer.toString(i) + "_C" + Integ...
...er.toString(j+1);
                        }

                        outBoundary.print(boundaryName);
                        outBoundary.print(" ");
                        outBoundary.print(corner1Name);
                        outBoundary.print(" ");
                        outBoundary.println(corner2Name);
                    }
                }

                // Add corners to corner.dat
                for(int i=0; i<convectionGroups.size(); i++){
                    ArrayList<GPoint2D<Double>> convectionGroupi =
                            convectionGroups.get(i);

                    int numCorners = convectionGroupi.size();

                    for(int j=0; j<numCorners; j++){
                        String corner1Name = "WR" + Integer.toString(ti...
...meOfForecast)
                            + "_B" + Integer.toString(i) + "_C" + Integ...
...er.toString(j);

                        GPoint2D<Double> cornerPt = convectionGroupi.ge...
...t(j);

                        outCorner.print(corner1Name);
                        outCorner.print(" ");
                        outCorner.print(cornerPt.getLatitude());
                        outCorner.print(" ");
                        outCorner.println(cornerPt.getLongitude());
                    }
                }

            }
```

```
            outRestriction.close();
            outRestrictBoundary.close();
            outBoundary.close();
            outCorner.close();
        }catch(IOException e){
            System.out.println("Could not write RAMS weather restrictio...
...n files!");
        }



        return this;
    }
}
```

FAPioWeatherReader.java

```
/*
 * FAPioWeatherReader.java
 *
 * Created on June 2, 2007, 2:27 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
import java.util.*;
import java.io.*;
import javax.imageio.*;
import java.awt.image.*;
/**
 *
 * @author Jeff
 */
public class FAPioWeatherReader {
    Thread tForecast;
    private Random random;

    // Thread class to read convection forecast
    private static class readForecast implements Runnable{
        private FAPioWeatherReader wr_;

        public readForecast(FAPioWeatherReader wr){
            wr_ = wr;
        }

        public void run(){
            wr_.readConvectionForecast();
            wr_.readConvectionTops();
            wr_.readWeibullParameters();
        }
    }

    /** Creates a new instance of FAPioWeatherReader */
    public FAPioWeatherReader() {
        random = new Random(Settings.randomSeed);
    }

    public FAPioWeatherReader convectionForecast(){
        tForecast = new Thread(new readForecast(this));
        tForecast.start();
        tForecast.setPriority(Thread.MAX_PRIORITY);
        return this;
    }
```

```
    public FAPioWeatherReader scenarioCapacityCalculator(){
        // Make sure finished reading forecast
        try{
            tForecast.join();
        }catch(InterruptedException e){
            System.out.println("Read of convection forecast interrupted...
...!");
        }

        calculateScenarioCapacity();

        return this;
    }

    /** Calculate minute-by-minute sector capacity for each weather
     *  scenario.
     */
    private FAPioWeatherReader calculateScenarioCapacity(){
        Airspace a = APData.airspace;
        Network n = APData.network;

        // Initialize weather scenario capacity and flow
        for(int i=0; i<n.getNumberWeatherScenarios(); i++){
            Scenario weatherScenario = n.getWeatherScenario(i);
            weatherScenario.initializeSectorCapacityAndFlow();
        }

        // Copying flight plan demand to weather scenarios
        APData.network.copyFlightPlanDemandToWeatherScenarios();

        /** Tops distribution
         *  Row is forecast (1 to 3)
         *  Column is observation (0 to 3)
         *  0 : N/A
         *  0 : < 25 kft
         *  1 : 25-31 kft
         *  2 : 31-37 kft
         *  3 : 37+ kft
         *
         *
        final double[][] TOPSDISTRIBUTION = {
            {0.03, 0.75, 0.21, 0.01, 0.00},
            {0.01, 0.39, 0.58, 0.02, 0.00},
            {0.01, 0.20, 0.67, 0.11, 0.01}
        };*/
        final double[][] TOPSDISTRIBUTIONCDF = {
```

```
            {0.03, 0.78, 0.99, 1.00, 1.00},
            {0.01, 0.40, 0.98, 1.00, 1.00},
            {0.01, 0.21, 0.88, 0.99, 1.00}
        };

        int buffermCount = 0;
        int polyboolCount = 0;
        int areaintCount = 0;


        ConvectionForecastDistribution cfd =
                APData.convectionForecastDistribution;
        for(int timeOfForecast:APData.convectionForecasts.keySet()){
            ConvectionForecast cf = APData.convectionForecasts.get(time...
...OfForecast);

            System.out.println("Number of convection forecast groups: "...
... +
                    Integer.toString(cf.noConvectionForecastGroups()));...
...           for(int n2 = 1; n2 <= cf.noConvectionForecastGroups(); n2++...
...){
                // Area of the forecast (nm^2) for this group
                double forecastArea = cf.getConvectionForecastArea(n2-1...
...);

                double weibullLambda10nmBuffer = cfd.getWeibullLambda10...
...nmBuffer(forecastArea);
                double weibullK10nmBuffer = cfd.getWeibullK10nmBuffer(f...
...orecastArea);

                double weibullLambda20nmBuffer = cfd.getWeibullLambda20...
...nmBuffer(forecastArea);
                double weibullK20nmBuffer = cfd.getWeibullK20nmBuffer(f...
...orecastArea);

                double weibullLambdaCentroidDist = cfd.getWeibullLambda...
...CentroidDist(forecastArea);
                double weibullKCentroidDist = cfd.getWeibullKCentroidDi...
...st(forecastArea);

                // Tops of the forecast (kft) for this group
                int forecastTops = cf.getConvectionForecastTops(n2-1);
                forecastTops--; // Adjust for MATLAB

                // Convection latitude longitude
                double[] latitudes = cf.getConvectionForecastGroupLatit...
...ude(n2-1);
```

```
                double[] longitudes = cf.getConvectionForecastGroupLong...
...itude(n2-1);

                for(int n3 = 1; n3 <= n.getNumberWeatherScenarios(); n3...
...++){
                    // Note that the first weather scenario is the fore...
...casted
                    // convection with no additional random component
                    boolean useForecast = n3 == 1;

                    Scenario weatherScenario = n.getWeatherScenario(n3-...
...1);

                    // Generate random tops
                    //Random random = new Random(Settings.randomSeed);
                    double rndTopsUni = random.nextDouble();
                    int randomTops = 0;
                    if (useForecast) {
                        randomTops = forecastTops;
                    } else {
                        for (int n4 = 0; n4 <= 4; n4++) {
                            if (TOPSDISTRIBUTIONCDF[forecastTops][n4] >...
... rndTopsUni) {
                                randomTops = n4 - 1;
                            }
                        }
                    }

                    // Do not consider if convection is not significant...
...                    if(randomTops <= 0) continue;

                    // Generate area bias, centroid-to-centroid dist, a...
...nd azimuth
                    double rndAreaBias10nm = weibullRand(weibullLambda1...
...0nmBuffer,
                            weibullK10nmBuffer) * forecastArea;
                    double rndAreaBias20nm = weibullRand(weibullLambda2...
...0nmBuffer,
                            weibullK20nmBuffer) * forecastArea;
                    double rndCentroidDist = weibullRand(weibullLambdaC...
...entroidDist,
                            weibullKCentroidDist);
                    double rndAzimuth = random.nextDouble() * 360.0D; /...
.../ In degrees

                    if(useForecast){
                        rndAreaBias10nm = 1.0D;
```

```
                        rndAreaBias20nm = 1.0D;
                        rndCentroidDist = 0.0D;
                        rndAzimuth = 0.0D;
                    }

                    // Move based on random centroid distance
                    double[][] coords = arrayToDblArray(latitudes, long...
...itudes);
                    double[][] rngAz = {{rndCentroidDist, rndAzimuth}};...
...                    double[][] reckOut = MatlabDriver.reckon2(coords, r...
...ngAz);

                    // Copy results
                    double[] latitudes2 = new double[reckOut.length];
                    double[] longitudes2 = new double[reckOut.length];
                    for(int i=0; i<reckOut.length; i++){
                        latitudes2[i] = reckOut[i][0];
                        longitudes2[i] = reckOut[i][1];
                    }

                    // Inflate/Deflate forecasted convection polygon to...
... math 10nm buffer bias
                    double radiusForecastArea = Math.sqrt(forecastArea/...
...Math.PI);
                    double radiusAreaBias10nm = Math.sqrt(rndAreaBias10...
...nm/Math.PI);
                    double radiusAreaBias20nm = Math.sqrt(rndAreaBias20...
...nm/Math.PI);


                    double[] lat10nm = null;
                    double[] lon10nm = null;
                    if(rndAreaBias10nm > forecastArea){
                        double[][] buffersize_nm = {{radiusAreaBias10nm...
... - radiusForecastArea}};
                        double[][] bufRes = MatlabDriver.buffermOut(rec...
...kOut, buffersize_nm);
                        buffermCount++;
                        lat10nm = new double[bufRes.length];
                        lon10nm = new double[bufRes.length];
                        for(int i=0; i<bufRes.length; i++){
                            lat10nm[i] = bufRes[i][0];
                            lon10nm[i] = bufRes[i][1];
                        }
                    }else{
                        double[][] buffersize_nm = {{radiusForecastArea...
... - radiusAreaBias10nm}};
```

```
                        double[][] bufRes = MatlabDriver.buffermIn(reck...
...Out, buffersize_nm);
                        buffermCount++;
                        lat10nm = new double[bufRes.length];
                        lon10nm = new double[bufRes.length];
                        for(int i=0; i<bufRes.length; i++){
                            lat10nm[i] = bufRes[i][0];
                            lon10nm[i] = bufRes[i][1];
                        }
                    }

                    double[] lat20nm = null;
                    double[] lon20nm = null;
                    if(rndAreaBias20nm > forecastArea){
                        double[][] buffersize_nm = {{radiusAreaBias20nm...
... - radiusForecastArea}};
                        double[][] bufRes = MatlabDriver.buffermOut(rec...
...kOut, buffersize_nm);
                        buffermCount++;
                        lat20nm = new double[bufRes.length];
                        lon20nm = new double[bufRes.length];
                        for(int i=0; i<bufRes.length; i++){
                            lat20nm[i] = bufRes[i][0];
                            lon20nm[i] = bufRes[i][1];
                        }
                    }else{
                        double[][] buffersize_nm = {{radiusForecastArea...
... - radiusAreaBias20nm}};
                        double[][] bufRes = MatlabDriver.buffermIn(reck...
...Out, buffersize_nm);
                        buffermCount++;
                        lat20nm = new double[bufRes.length];
                        lon20nm = new double[bufRes.length];
                        for(int i=0; i<bufRes.length; i++){
                            lat20nm[i] = bufRes[i][0];
                            lon20nm[i] = bufRes[i][1];
                        }
                    }

                    if(lat10nm.length == 0 || lat20nm.length == 0) cont...
...inue;

                    // Get the centroid of the randomly generated conve...
...ction
                    double[][] coord = arrayToDblArray(lat10nm, lon10nm...
...);
                    double[][] mnOut = MatlabDriver.meanm2(coord);
```

```
                    GPoint2D<Double> pt = new GPoint2D<Double>(
                            mnOut[0][0], mnOut[0][1]);

                    // Store the random convection generation results
                    int startTimeMin = (timeOfForecast + 1)*60;
                    int endTimeMin = startTimeMin + 60;
                    endTimeMin = Math.min(1439, endTimeMin);
                    weatherScenario.addRandomConvection(pt, rndAreaBias...
...10nm,
                            lat10nm, lon10nm, rndAreaBias20nm, lat20nm,...
... lon20nm,
                            randomTops, startTimeMin, endTimeMin);

                    int randomTopsNum = 0;
                    if(randomTops == 0) randomTopsNum = 0;
                    else if(randomTops == 1) randomTopsNum = 310;
                    else if(randomTops == 2) randomTopsNum = 370;
                    else randomTopsNum = 999;

                    // Preprocessing of lat/lon to exclude centers
                    double minConvectionLat = min(lat20nm);
                    double maxConvectionLat = max(lat20nm);
                    double minConvectionLon = min(lon20nm);
                    double maxConvectionLon = max(lon20nm);

                    // Iterate through each sector and update capacity
                    for(String centerS:FAPio.writeCenters){
                        ACenter center = a.getCenter(centerS);

                        // Quick check that convection is in area of ce...
...nter
                        double minCenterLat = center.getMinimumLat();
                        double maxCenterLat = center.getMaximumLat();
                        double minCenterLon = center.getMinimumLon();
                        double maxCenterLon = center.getMaximumLon();

                        if((minConvectionLat < minCenterLat ||
                                minConvectionLat > maxCenterLat) &&
                                (maxConvectionLat < minCenterLat ||
                                maxConvectionLat > maxCenterLat)){
                            continue;
                        }

                        if((minConvectionLon < minCenterLon ||
                                minConvectionLon > maxCenterLon) &&
                                (maxConvectionLon < minCenterLon ||
                                maxConvectionLon > maxCenterLon)){
```

```
                        continue;
                    }


                    Iterator<ASector> e = center.getSectorValueIter...
...ator();
                    while(e.hasNext()){
                        ASector sector = e.next();

                        // Quick check that convection is in area o...
...f sector
                        double minSectorLat = sector.getMinimumLat(...
...);
                        double maxSectorLat = sector.getMaximumLat(...
...);
                        double minSectorLon = sector.getMinimumLon(...
...);
                        double maxSectorLon = sector.getMaximumLon(...
...);

                        if((minConvectionLat < minSectorLat ||
                                minConvectionLat > maxSectorLat) &&...
...                                 (maxConvectionLat < minSectorLat ||...
...                                 maxConvectionLat > maxSectorLat)){
                            continue;
                        }

                        if((minConvectionLon < minSectorLon ||
                                minConvectionLon > maxSectorLon) &&...
...                                 (maxConvectionLon < minSectorLon ||...
...                                 maxConvectionLon > maxSectorLon)){
                            continue;
                        }

                        // Only update capacity of sectors consider...
...ed in this analysis
                        int arrayIndex = sector.getArrayIndex();
                        if(arrayIndex < 0) continue;

                        double totalVolume = 0.0D;
                        double totalImpactedVolume = 0.0D;
                        int numIntersections = 0;

                        for(ASectorModule sectorModule:sector.getMo...
...dules()){
                            double floor = sectorModule.getFloor();...
...                             double ceiling = sectorModule.getCeilin...
```

```
...g();

                                if(floor > randomTopsNum) continue;

                                double[] latNodes = sectorModule.getNod...
...eLatitudes();
                                double[] lonNodes = sectorModule.getNod...
...eLongitudes();

                                // Module height
                                final double FEETTONAUTICALMILES = 0.00...
...0164578834D;
                                double moduleHeight = (ceiling - floor)...
... * 100; // ft
                                double moduleHeightNm = moduleHeight * ...
...FEETTONAUTICALMILES;

                                // Area of module
                                double moduleArea = sectorModule.getAre...
...a();
                                if(moduleArea <= 0.0D){
                                    double[][] coord2 = arrayToDblArray...
...(latNodes, lonNodes);
                                    double[][] aintOut = MatlabDriver.a...
...reaintNm(coord2);
                                    moduleArea = aintOut[0][0];
                                    sectorModule.setArea(moduleArea);
                                }

                                // Volume of module
                                double moduleVolume = moduleArea * modu...
...leHeightNm;
                                totalVolume += moduleVolume;

                                // Find intersection of module and fore...
...casted convection
                                double[] lat_inter = null;
                                double[] lon_inter = null;
                                double[][] pbOut = null;
                                if(floor > 180.0D){
                                    pbOut = MatlabDriver.polyboolInters...
...ection(
                                            arrayToDblArray(latNodes, l...
...onNodes),
                                            arrayToDblArray(lat20nm, lo...
...n20nm));
                                            polyboolCount++;
```

```
                            }else{
                                pbOut = MatlabDriver.polyboolInters...
...ection(
                                        arrayToDblArray(latNodes, l...
...onNodes),
                                        arrayToDblArray(lat10nm, lo...
...n10nm));
                                        polyboolCount++;
                            }

                            lat_inter = new double[pbOut.length];
                            lon_inter = new double[pbOut.length];
                            for(int i=0; i<pbOut.length; i++){
                                lat_inter[i] = pbOut[i][0];
                                lon_inter[i] = pbOut[i][1];
                            }

                            if(lat_inter.length == 0) continue;
                            else numIntersections++;

                            double[][] aintOut = MatlabDriver.areai...
...ntNm(
                                    arrayToDblArray(lat_inter, lon_...
...inter));
                            areaintCount++;

                            // May be more than one area
                            double intersectionArea = 0.0D;
                            for(int i=0; i<aintOut.length; i++){
                                double[] aintOut2 = aintOut[i];
                                for(int j=0; j<aintOut2.length; j++...
...){
                                    intersectionArea += aintOut2[j]...
...;
                                }
                            }

                            double intersectionVolume = intersectio...
...nArea *
                                    moduleHeightNm;

                            totalImpactedVolume += intersectionVolu...
...me;
                        }

                        // Don't update if no intersections are fou...
...nd
```

```
                            if(numIntersections == 0) continue;

                            // Multiplier for capacity
                            double capacityMultiplier = Math.max(0.0D,
                                    (totalVolume - totalImpactedVolume)...
.../totalVolume);

                            // Update capacity for scenario
                            // Note that capacity time periods are in
                            // 15 minutes increments
                            double startTimeMin15_ = (double)startTimeM...
...in/15.0D
                                    - startTimeMin % 15;
                            int startTimeMin15 = (int)Math.round(startT...
...imeMin15_);

                            double endTimeMin15_ = (double)endTimeMin/1...
...5.0D
                                    - endTimeMin % 15;
                            int endTimeMin15 = (int)Math.round(endTimeM...
...in15_);

                            for(int t = startTimeMin15; t <= endTimeMin...
...15; t++){
                                double currentCapacity = weatherScenari...
...o.getSectorCapacity15Min(
                                        arrayIndex, t);
                                double revisedCapacity = currentCapacit...
...y * capacityMultiplier;

                                weatherScenario.setSectorCapacity15Min(...
...arrayIndex, t, revisedCapacity);
                            }
                        }
                    }
                }
            }

        }

        System.out.println(buffermCount);
        System.out.println(polyboolCount);
        System.out.println(areaintCount);

        return this;
    }
```

```
    private static double max(double[] t) {
        double maximum = t[0];   // start with the first value
        for (int i=1; i<t.length; i++) {
            if (t[i] > maximum) {
                maximum = t[i];   // new maximum
            }
        }
        return maximum;
    }

    private static double min(double[] t) {
        double minimum = t[0];   // start with the first value
        for (int i=1; i<t.length; i++) {
            if (t[i] > minimum) {
                minimum = t[i];   // new maximum
            }
        }
        return minimum;
    }

    private double weibullRand(double a, double b){
        //Random random = new Random(Settings.randomSeed);
        double rand = random.nextDouble();

        return a * Math.pow((-1 * Math.log(rand)),1/b);
    }

    private int convertJavaImgToMatlabImg(int rgb){
        final int[] matlabImgInt = {1,2,4,8,11,12};
        final int[] javaImgInt = {-16777216,-13255115,-256,-1120040,-53...
...85490,
            -4934476};

        for(int i=0; i<matlabImgInt.length; i++){
            if(rgb == javaImgInt[i]){
                return matlabImgInt[i];
            }
        }

        return 0;
    }


    private int[][] readNcwfImage(String fileName){
        try{
            File f = new File(fileName);
```

```
            BufferedImage ncwfImage = ImageIO.read(f);


            int imgHeight = ncwfImage.getHeight();
            int imgWidth = ncwfImage.getWidth();

            int[][] cdata = new int[imgHeight][imgWidth];
            for(int i=0; i<imgHeight; i++){
                for(int j=0; j<imgWidth; j++){
                    int val = convertJavaImgToMatlabImg(ncwfImage.getRG...
...B(j,i));
                    cdata[i][j] = val;
                }
            }

            return cdata;

        }catch(IOException e){
            System.out.println("Could not read image from: " + fileName...
...);
            return null;
        }
    }

    private void printCdata(int[][] cdata){
        for(int k=1; k<=50; k++){
            int k_cnt = 0;
            for(int i=0; i<cdata.length; i++){
                for(int j=0; j<cdata[0].length; j++){
                    int val = cdata[i][j];

                    if(val == k) k_cnt++;
                }
            }
            System.out.print(k);System.out.print(" : ");
            System.out.println(k_cnt);
        }

    }

    private FAPioWeatherReader readConvectionTops(){
        String fileName = Settings.convectionTopsFilename;

        try{
            BufferedReader d = new BufferedReader(new FileReader(fileNa...
...me));
```

```
            String line = null;
            while((line=d.readLine()) != null){
                String[] dataFields = line.split(",");

                if(dataFields.length < 6) continue;

                int hour = Integer.parseInt(dataFields[0]);
                int convectionGroupNum = Integer.parseInt(dataFields[1]...
...);
                double lat = Double.parseDouble(dataFields[2]);
                double lon = Double.parseDouble(dataFields[3]);
                int forecastTops = Integer.parseInt(dataFields[4]);
                int observedTops = Integer.parseInt(dataFields[5]);

                GPoint2D<Double> pt = new GPoint2D<Double>(lon, lat);

                int hIdxStart = hour == 1 ? 0 : hour;

                for(int i=hIdxStart; i<=hour+1; i++){
                    int timeOfForecastHour = i;
                    if(!APData.convectionForecasts.containsKey(
                            timeOfForecastHour)) continue;

                    ConvectionForecast cf = APData.convectionForecasts....
...get(
                            timeOfForecastHour);

                    // Add forecasted tops
                    if(convectionGroupNum == 1){
                        cf.addConvectionForecastTops(0, forecastTops, p...
...t);
                    }else{
                        for(int n=1; n <= cf.noConvectionForecastGroups...
...(); n++){
                            // Get current distance from convection gro...
...up to tops
                            GPoint2D<Double> forecastCentroid = cf.getC...
...onvectionForecastGroupCentroid(n-1);
                            GPoint2D<Double> topsCentroid = cf.getConve...
...ctionForecastTopsLocation(n-1);

                            Mapping m = new Mapping(forecastCentroid.ge...
...tLatitude(),
                                    forecastCentroid.getLongitude(),
                                    topsCentroid.getLatitude(),
                                    topsCentroid.getLongitude());
                            //m.calculateDistance();
```

```
                            double dist1nm = m.getDistance();

                            m = new Mapping(forecastCentroid.getLatitud...
...e(),
                                    forecastCentroid.getLongitude(),
                                    lat,
                                    lon);
                            //m.calculateDistance();
                            double dist2nm = m.getDistance();

                            if(dist2nm < dist1nm){
                                cf.addConvectionForecastTops(n-1, forec...
...astTops, pt);
                            }
                        }
                    }
                    // Add observed tops
                    cf.addObservedConvectionTops(observedTops, pt);
                }
            }

            d.close();
        }catch(IOException e){
            System.out.println("Could not open convection tops file: " ...
...+ fileName);
        }

        return this;
    }

    private FAPioWeatherReader readWeibullParameters(){
        String fileName = Settings.ncwfWeibullParametersFilename;

        try{
            BufferedReader d = new BufferedReader(new FileReader(fileNa...
...me));

            String line = null;
            while((line = d.readLine()) != null){
                if(line.startsWith("ForecastArea")) continue;

                String[] dataFields = line.split(",");
                if(dataFields.length < 9) continue;

                double forecastArea = Double.parseDouble(dataFields[0])...
...;
                double weibullLambda0nmBuffer = Double.parseDouble(data...
```

```
...Fields[1]);
                double weibullK0nmBuffer = Double.parseDouble(dataField...
...s[2]);
                double weibullLambda10nmBuffer = Double.parseDouble(dat...
...aFields[3]);
                double weibullK10nmBuffer = Double.parseDouble(dataFiel...
...ds[4]);
                double weibullLambda20nmBuffer = Double.parseDouble(dat...
...aFields[5]);
                double weibullK20nmBuffer = Double.parseDouble(dataFiel...
...ds[6]);
                double weibullLambdaCentroidDist = Double.parseDouble(d...
...ataFields[7]);
                double weibullKCentroidDist = Double.parseDouble(dataFi...
...elds[8]);

                APData.convectionForecastDistribution.addData(forecastA...
...rea,
                    weibullLambda0nmBuffer,
                    weibullK0nmBuffer,
                    weibullLambda10nmBuffer,
                    weibullK10nmBuffer,
                    weibullLambda20nmBuffer,
                    weibullK20nmBuffer,
                    weibullLambdaCentroidDist,
                    weibullKCentroidDist);
            }

            d.close();
        }catch(IOException e){
            System.out.println("Could not read Weibull parameters from ...
...file: " + fileName);
        }

        return this;
    }

    private FAPioWeatherReader readConvectionForecast(){
        // Iterate through all *.gif files in the convection forecast d...
...irectory
        File fileGif = new File(Settings.convectionForecastDirectory);
        String[] fileList = fileGif.list();

        for(int i=0; i<fileList.length; i++){
            // Check that this is a gif file
            if(!fileList[i].endsWith(".gif")) continue;
```

```
            // Only consider NCWF forecasts
            if(!fileList[i].startsWith("ncwfp_ncwdp")) continue;

            String fileName = Settings.convectionForecastDirectory + fi...
...leList[i];

            // Get the time of the forecast
            // Example: ncwfp_ncwdp_national_20050727_2100_1hr_cutoff1....
...gif
            String[] fileNames = fileList[i].split("_");
            int timeOfForecast = Integer.parseInt(
                    fileNames[fileNames.length - 3].substring(0,2));

            if(timeOfForecast < Settings.analysisTimeFilterStart
                    || timeOfForecast > Settings.analysisTimeFilterEnd)...
... continue;
            System.out.println("Using the forecast at hour " +
                    Integer.toString(timeOfForecast) + "...");

            // Call function to read convection info
            ConvectionForecast cf = getNcwfRtvsConvection(fileName);

            // Store the result
            APData.convectionForecasts.put(timeOfForecast, cf);
        }

        return this;
    }

    private ConvectionForecast getNcwfRtvsConvection(String fileName){
        //% This needs to return the forecast and observed convection l...
...ocations
        //
        //% Function to read NCWF forecast and observed convection from...
... RTVS website
        //% at : http://rtvs.noaa.gov/conv/2001/display/
        //%
        //% cdata_conv: processed image for convective forecast (see cd...
...ata_ncwf)
        //% cdata_ncwf: just the forecast
        //% cdata_ncwf2: colored version of cdata_ncwf for validation
        //% cdata_trans_lat: latitude of each pixel in the image
        //% cdata_trans_lon: longitude of each pixel in the image
        //% avg_ncwf: average location (lat,lon,count) of each convecti...
...on forecast
        //% conv_stats: statistics on the distance and size of observed...
... convection
```

```
        //%             as compared to forecasted convection.
        //%

        // Convert cell to geographic area (nm^2)
        final double AREAPERPIXEL = 9.2365;

        // Create convection forecast object
        ConvectionForecast cf = new ConvectionForecast();

        int img = 1;

        int[][] cdata = readNcwfImage(fileName);

        int num_rows = cdata.length;
        int num_cols = cdata[0].length;

        //% To recover the image:
        //% >> image(cdata); colormap(colourmap_2); axis image;
        //%
        //% Colour Values
        //% 0: White
        //% 1: Black (Outline, States, Lettering)
        //% 2: Green (Convection)
        //% 4: Yellow (NCWF Forecast)
        //% 6: Red (Lettering)
        //% 8: Light Brown (Land)
        //% 11: Blue (Water)
        //% 12: Grey (ARTCC Boundaries)
        //%
        //% Points to match image locations to geographic latitude/long...
...itude
        //% lat_map = [48.38; 45.0014; 41.9938; 40.998; 36.999; 31.3324...
...; 25.11];
        //% lat_img = [32; 113; 181; 203; 288; 401; 519];
        //% lon_map = [-124.725; -111.0544; -114.041; -111.046; -109.04...
...4; -109.0494; -81.1];
        //% lon_img = [201;385;345;385;412;412;789];

        final int BLACK_COLOR_IDX = 1;
        final int CONVECTION_COLOR_IDX = 2;
        final int NCWF_COLOR_IDX = 4;
        final int BUFFER_COLOR_IDX = 5;
        final int LAND_COLOR_IDX = 8;
        final int ARTCC_COLOR_IDX = 12;
        final int MIN_IMAGE_X = 160;  // Exclude the legend

        // Copy of the original image
```

```
        int[][] cdata_orig = intArrayCopy(cdata);

        // NCWF cells processed for convection
        int[][] cdata_conv = intArrayCopy(cdata);
        for(int k=1; k<=3; k++){
            for(int i=5; i<= 550-6; i++){
                for(int j=MIN_IMAGE_X+4; j<=1000-6; j++){
                    // Check for ARTCC boundaries
                    if(k == 1 && cdata_conv[i][j] == ARTCC_COLOR_IDX){
                        if(cdata_conv[i+1][j] == NCWF_COLOR_IDX ||
                                cdata_conv[i-1][j] == NCWF_COLOR_IDX ||...
...                                cdata_conv[i][j+1] == NCWF_COLOR_IDX ||...
...                                cdata_conv[i][j-1] == NCWF_COLOR_IDX){
                            if(cdata_conv[i+1][j] == LAND_COLOR_IDX ||
                                    cdata_conv[i-1][j] == LAND_COLOR_ID...
...X ||
                                    cdata_conv[i][j+1] == LAND_COLOR_ID...
...X ||
                                    cdata_conv[i][j-1] == LAND_COLOR_ID...
...X){
                                cdata_conv[i][j] = LAND_COLOR_IDX;
                            }else if(cdata_conv[i+1][j] == BLACK_COLOR_...
...IDX ||
                                    cdata_conv[i-1][j] == BLACK_COLOR_I...
...DX ||
                                    cdata_conv[i][j+1] == BLACK_COLOR_I...
...DX ||
                                    cdata_conv[i][j-1] == BLACK_COLOR_I...
...DX){
                                cdata_conv[i][j] = BLACK_COLOR_IDX;
                            }else{
                                cdata_conv[i][j] = NCWF_COLOR_IDX;
                            }
                        }
                    }

                    // Check for black outlines
                    if(cdata_conv[i][j] == BLACK_COLOR_IDX){
                        // Single black lines
                        if(cdata_conv[i+1][j] == NCWF_COLOR_IDX &&
                                cdata_conv[i-1][j] == NCWF_COLOR_IDX){
                            cdata_conv[i][j] = NCWF_COLOR_IDX;
                        }
                        if(cdata_conv[i][j+1] == NCWF_COLOR_IDX &&
                                cdata_conv[i][j-1] == NCWF_COLOR_IDX){
                            cdata_conv[i][j] = NCWF_COLOR_IDX;
                        }
```

```
                        // Double black lines
                        if(cdata_conv[i+2][j] == NCWF_COLOR_IDX &&
                                cdata_conv[i+1][j] == BLACK_COLOR_IDX &...
...&
                                cdata_conv[i-1][j] == NCWF_COLOR_IDX){
                            cdata_conv[i][j] = NCWF_COLOR_IDX;
                            cdata_conv[i+1][j] = NCWF_COLOR_IDX;
                        }
                        if(cdata_conv[i][j+2] == NCWF_COLOR_IDX &&
                                cdata_conv[i][j+1] == BLACK_COLOR_IDX &...
...&
                                cdata_conv[i][j-1] == NCWF_COLOR_IDX){
                            cdata_conv[i][j] = NCWF_COLOR_IDX;
                            cdata_conv[i][j+1] = NCWF_COLOR_IDX;
                        }
                    }

                    // Change convection to NCWF if adjacent
                    if(cdata_conv[i][j] == CONVECTION_COLOR_IDX){
                        if(cdata_conv[i+1][j] == NCWF_COLOR_IDX ||
                                cdata_conv[i-1][j] == NCWF_COLOR_IDX ||...
...                                cdata_conv[i][j+1] == NCWF_COLOR_IDX ||...
...                                cdata_conv[i][j-1] == NCWF_COLOR_IDX){
                            cdata_conv[i][j] = NCWF_COLOR_IDX;
                        }
                    }
                }
            }
        }

        // Group of NCWF cells
        int[][] cdata_ncwf = new int[cdata.length][cdata[0].length];
        int convection_group_num = 0;
        for(int k=1; k <= 50; k++){
            for(int i=1; i < num_rows - 1; i++){
                for(int j=MIN_IMAGE_X; j < num_cols - 1; j++){
                    if(cdata_conv[i][j] == NCWF_COLOR_IDX){
                        if(cdata_ncwf[i+1][j] > 0){
                            cdata_ncwf[i][j] = cdata_ncwf[i+1][j];
                        }else if(cdata_ncwf[i-1][j] > 0){
                            cdata_ncwf[i][j] = cdata_ncwf[i-1][j];
                        }else if(cdata_ncwf[i][j+1] > 0){
                            cdata_ncwf[i][j] = cdata_ncwf[i][j+1];
                        }else if(cdata_ncwf[i][j-1] > 0){
                            cdata_ncwf[i][j] = cdata_ncwf[i][j-1];
                        }else{
```

```
                    convection_group_num++;
                    cdata_ncwf[i][j] = convection_group_num;
                }
            }

            if(cdata_ncwf[i][j] > cdata_ncwf[i+1][j] &&
                    cdata_ncwf[i+1][j] > 0){
                cdata_ncwf[i][j] = cdata_ncwf[i+1][j];
            }else if(cdata_ncwf[i][j] > cdata_ncwf[i-1][j] &&
                    cdata_ncwf[i-1][j] > 0){
                cdata_ncwf[i][j] = cdata_ncwf[i-1][j];
            }else if(cdata_ncwf[i][j] > cdata_ncwf[i][j+1] &&
                    cdata_ncwf[i][j+1] > 0){
                cdata_ncwf[i][j] = cdata_ncwf[i][j+1];
            }else if(cdata_ncwf[i][j] > cdata_ncwf[i][j-1] &&
                    cdata_ncwf[i][j-1] > 0){
                cdata_ncwf[i][j] = cdata_ncwf[i][j-1];
            }
        }
    }
}

// Remove groups with no convection
for(int k2=0; k2 <= convection_group_num; k2++){
    int k = convection_group_num - k2;
    int count = 0;
    for(int i=0; i < num_rows; i++){
        for(int j=MIN_IMAGE_X; j < num_cols; j++){
            if(cdata_ncwf[i][j] == k){
                count++;
                break;
            }
        }
        if(count > 0) break;
    }

    // If no cells found decrement higher group numbers
    if(count == 0){
        for(int i = 0; i < num_rows; i++){
            for(int j = MIN_IMAGE_X; j < num_cols; j++){
                if(cdata_ncwf[i][j] > k)
                    cdata_ncwf[i][j] = cdata_ncwf[i][j] - 1;
            }
        }
    }
}
```

```
        // Find the new highest convection number
        int revised_convection_group_num = 0;
        for(int k=1; k <= convection_group_num; k++){
            int count = 0;
            for(int i=0; i < num_rows; i++){
                for(int j=MIN_IMAGE_X; j < num_cols; j++){
                    if(cdata_ncwf[i][j] == k){
                        count++;
                        break;
                    }
                }
                if(count > 0) break;
            }

            if(count == 0){
                revised_convection_group_num = k-1;
                break;
            }
        }
        convection_group_num = revised_convection_group_num;

        // Create polygon of forecast convection boundary
        for(int k=1; k <= convection_group_num; k++){
            ArrayList<Double> lat_l = new ArrayList<Double>();
            ArrayList<Double> lon_l = new ArrayList<Double>();
            double forecast_area = 0.0D;
            double centroid_lat = 0.0D;
            double centroid_lon = 0.0D;
            int n = 0;
            for(int i=0; i < num_rows; i++){
                for(int j = MIN_IMAGE_X; j < num_cols; j++){
                    if(cdata_ncwf[i][j] == k){
                        double iD = (double)i;
                        double jD = (double)j;
                        //double[] lat_l = {iD+0.5D,iD+0.5D,iD-0.5D,iD-...
...0.5D};
                        //double[] lon_l = {jD-0.5D,jD+0.5D,jD+0.5D,jD-...
...0.5D};
                        lat_l.add(iD+0.5D);lat_l.add(iD+0.5D);
                        lat_l.add(iD-0.5D);lat_l.add(iD-0.5D);
                        lat_l.add(Double.NaN);
                        lon_l.add(jD-0.5D);lon_l.add(jD+0.5D);
                        lon_l.add(jD+0.5D);lon_l.add(jD-0.5D);
                        lon_l.add(Double.NaN);

                        forecast_area += AREAPERPIXEL;
                        n++;
```

```
                    centroid_lat += i;
                    centroid_lon += j;
                }
            }
        }

        // Create input suitable for MATLAB component
        double[][] pbInput = arrayListsToDblArray(lat_l,lon_l);
        double[][] pbOutput = MatlabDriver.polyboolUnion(pbInput,pb...
...Input);

        double[] latu_k = new double[pbOutput.length];
        double[] lonu_k = new double[pbOutput.length];

        // Calculate latitude/longitude
        for(int i=0; i<pbOutput.length; i++){
            double latu_ki = pbOutput[i][0];
            double lonu_ki = pbOutput[i][1];

            latu_k[i] = (1.5973e-11)*Math.pow(latu_ki,4)-(1.2599e-8...
...)*Math.pow(latu_ki,3)-
                (1.2157e-5)*Math.pow(latu_ki,2)-0.039851*latu_ki+49...
....669;

            lonu_k[i] = (-3.7111e-9)*Math.pow(lonu_ki,3)+(4.8618e-6...
...)*Math.pow(lonu_ki,2)+
                0.072428*lonu_ki-139.45;
        }

        // Calculate the centroid
        centroid_lat = centroid_lat / n;
        centroid_lon = centroid_lon / n;
        centroid_lat = (1.5973e-11)*Math.pow(centroid_lat,4)-(1.259...
...9e-8)*Math.pow(centroid_lat,3)-
            (1.2157e-5)*Math.pow(centroid_lat,2)-0.039851*centroid_...
...lat+49.669;
        centroid_lon = (-3.7111e-9)*Math.pow(centroid_lon,3)+(4.861...
...8e-6)*Math.pow(centroid_lon,2)+
            0.072428*centroid_lon-139.45;

        // Store the results
        cf.addConvectionForecastGroup(lonu_k, latu_k, centroid_lon,...
... centroid_lat
                , forecast_area);
    }

    // Latitude/Longitude transformation of cell indices
```

```
        double[][] cdata_trans_lat = new double[num_rows][num_cols];
        double[][] cdata_trans_lon = new double[num_rows][num_cols];
        int num_convection_cells = 0;
        double observed_convection_area = 0;
        for(int i=0; i<num_rows; i++){
            for(int j=MIN_IMAGE_X; j<num_cols; j++){
                cdata_trans_lat[i][j] = (1.5973e-11)*Math.pow(i,4)-(1.2...
...599e-8)*Math.pow(i,3)-
                    (1.2157e-5)*Math.pow(i,2)-0.039851*i+49.669;
                cdata_trans_lon[i][j] = (-3.7111e-9)*Math.pow(j,3)+(4.8...
...618e-6)*Math.pow(j,2)+
                    0.072428*j-139.45;

                // Calculate number of convection cells
                if(cdata_orig[i][j] == CONVECTION_COLOR_IDX){
                    num_convection_cells++;
                    observed_convection_area += AREAPERPIXEL;
                }
            }
        }

        // Create polygon for observed convection
        double[] lat_conv = new double[num_convection_cells*5];
        double[] lon_conv = new double[num_convection_cells*5];
        int k=0;
        for(int i=0; i<num_rows; i++){
            for(int j=MIN_IMAGE_X; j<num_cols; j++){
                if(cdata_orig[i][j] == CONVECTION_COLOR_IDX){
                    lat_conv[k] = (double)i - 0.5D;
                    lon_conv[k] = (double)j - 0.5D;
                    k++;
                    lat_conv[k] = (double)i - 0.5D;
                    lon_conv[k] = (double)j + 0.5D;
                    k++;
                    lat_conv[k] = (double)i + 0.5D;
                    lon_conv[k] = (double)j + 0.5D;
                    k++;
                    lat_conv[k] = (double)i + 0.5D;
                    lon_conv[k] = (double)j - 0.5D;
                    k++;
                    lat_conv[k] = Double.NaN;
                    lon_conv[k] = Double.NaN;
                    k++;
                }
            }
        }
```

```
        double[][] coord_conv = arrayToDblArray(lat_conv, lon_conv);
        double[][] pbOutput = MatlabDriver.polyboolUnion(coord_conv, co...
...ord_conv);

        lat_conv = new double[pbOutput.length];
        lon_conv = new double[pbOutput.length];
        for(int i=0; i<pbOutput.length; i++){
            lat_conv[i] = (1.5973e-11)*Math.pow(pbOutput[i][0],4)-(1.25...
...99e-8)*Math.pow(pbOutput[i][0],3)-
                (1.2157e-5)*Math.pow(pbOutput[i][0],2)-0.039851*pbOutpu...
...t[i][0]+49.669;
            lon_conv[i] = (-3.7111e-9)*Math.pow(pbOutput[i][1],3)+(4.86...
...18e-6)*Math.pow(pbOutput[i][1],2)+
                0.072428*pbOutput[i][1]-139.45;
        }
        double[][] bmInput = arrayToDblArray(lat_conv, lon_conv);

        // Use the MATLAB buffer function for 10 nm and 20 nm
        double[][] bufferdist10 = {{10.0D}};
        double[][] bufferdist20 = {{20.0D}};
        double[][] bmOutput10 = MatlabDriver.buffermOut(bmInput,bufferd...
...ist10);
        double[][] bmOutput20 = MatlabDriver.buffermOut(bmInput,bufferd...
...ist20);

        double[] lat_conv_10nm_buffer = new double[bmOutput10.length];
        double[] lon_conv_10nm_buffer = new double[bmOutput10.length];
        for(int i=0; i<bmOutput10.length; i++){
            lat_conv_10nm_buffer[i] = bmOutput10[i][0];
            lon_conv_10nm_buffer[i] = bmOutput10[i][1];
        }

        double[] lat_conv_20nm_buffer = new double[bmOutput20.length];
        double[] lon_conv_20nm_buffer = new double[bmOutput20.length];
        for(int i=0; i<bmOutput20.length; i++){
            lat_conv_20nm_buffer[i] = bmOutput20[i][0];
            lon_conv_20nm_buffer[i] = bmOutput20[i][1];
        }

        // Store the results
        cf.addObservedConvection(lat_conv,lon_conv,lat_conv_10nm_buffer...
...,
                lon_conv_10nm_buffer,lat_conv_20nm_buffer,
                lon_conv_20nm_buffer);

        return cf;
    }
```

```
    private static double[][] arrayToDblArray(double[] a, double[] b){
        double[][] arrayToReturn = new double[a.length][2];

        for(int i=0; i<a.length; i++){
            arrayToReturn[i][0] = a[i];
            arrayToReturn[i][1] = b[i];
        }

        return arrayToReturn;
    }

    private static double[][] arrayListsToDblArray(ArrayList<Double> a,...
...            ArrayList<Double> b){
        double[][] arrayToReturn = new double[a.size()][2];

        for(int i=0; i<a.size(); i++){
            arrayToReturn[i][0] = a.get(i);
            arrayToReturn[i][1] = b.get(i);
        }

        return arrayToReturn;
    }

    /** Copy an array (no reference) */
    private int[][] intArrayCopy(int[][] src){
        // Return null if src is zero length
        if(src.length == 0) return null;

        int[][] dest = new int[src.length][src[0].length];

        for(int i=0; i<src.length; i++){
            for(int j=0; j<src[i].length; j++){
                dest[i][j] = src[i][j];
            }
        }

        return dest;
    }

//    public static void ncwfImageAnalysis(int[][] cdata){
//        HashMap<Integer,Integer> valCnt = new HashMap<Integer,Integer...
...>(
//                Settings.hashMapDefaultInitialCapacity,
//                Settings.hashMapLoadFactor);
//        for(int i=0; i<cdata.length; i++){
//            for(int j=0; j<cdata[i].length; j++){
```

```
//                int val = cdata[i][j];
//                if(!valCnt.containsKey(val)){
//                    valCnt.put(val,1);
//                }else{
//                    int cnt = valCnt.get(val);
//                    cnt++;
//                    valCnt.put(val,cnt);
//                }
//            }
//        }
//
//        Iterator<Integer> e = valCnt.keySet().iterator();
//        while(e.hasNext()){
//            int i = e.next();
//            System.out.print(i); System.out.print(" : ");
//            System.out.println(valCnt.get(i));
//        }
//
//    }

}
```

FAPioWindReader.java

```
/*
 * FAPioWindReader.java
 *
 * Created on May 26, 2007, 1:46 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author Jeff
 */
import java.util.*;
import java.io.*;
public class FAPioWindReader {
    private static final double MIN_TO_DEG = (1.0D/60.0D);

    private Wind wind_;
    private String windStationFileName_;
    private String windDataFileName_;

    /** Creates a new instance of FAPioWindReader */
    public FAPioWindReader(Wind wind, String windStationFileName,
            String windDataFileName) {
        wind_ = wind;;
        windStationFileName_ = windStationFileName;
        windDataFileName_ = windDataFileName;
    }

    public FAPioWindReader readWindStations() throws IOException{
        BufferedReader d = new BufferedReader(new FileReader(windStatio...
...nFileName_));
        String line;

        while((line = d.readLine()) != null){
            // Spaces
            int lastSpace = line.lastIndexOf(APData.M);
            int middleSpace = line.lastIndexOf(APData.SPACE, lastSpace ...
...- 1);
            int firstSpace = line.lastIndexOf(APData.SPACE, middleSpace...
... - 1);

            // Degrees (d) and minutes (m) indicators
            int lastD = line.lastIndexOf(APData.D);
            int firstD = line.lastIndexOf(APData.D, lastD - 1);
```

```
            int firstM = line.lastIndexOf(APData.M, lastSpace - 1);

            // If error check format of station_locations.txt
            String station = line.substring(0,3);
            String latitudeDeg = line.substring(firstSpace+1, firstD);
            String latitudeMin = line.substring(firstD+1, firstM);
            String longitudeDeg = line.substring(middleSpace+1, lastD);...
...            String longitudeMin = line.substring(lastD+1, lastSpace);

            double stationLatitude = Double.parseDouble(latitudeDeg);  ...
...
            stationLatitude += Math.signum(stationLatitude)*MIN_TO_DEG
                    *Double.parseDouble(latitudeMin);

            double stationLongitude = Double.parseDouble(longitudeDeg);...
...            stationLongitude += Math.signum(stationLongitude)*MIN_TO_DE...
...G
                    *Double.parseDouble(longitudeMin);
                    //MIN_TO_DEG*Double.parseDouble(latitudeMin);
            wind_.addWindStation(station, stationLatitude, stationLongi...
...tude);
        }

        line = null;

        d.close();

        return this;
    }

    public FAPioWindReader readWindData() throws IOException{
        BufferedReader d = new BufferedReader(new FileReader(Settings.w...
...indData));

        // Reading wind direction and speed
        //final int[] flightLevel = {30, 60, 90, 120, 180, 240, 300, 34...
...0, 390};
        final int[] startIndex = {5, 10, 18, 26, 34, 42, 50, 57, 64};
        final int[] endIndex = {9, 14, 22, 30, 38, 46, 54, 61, 68};

        boolean readThisData = false;
        int startTime = 0;
        String line;

        while((line = d.readLine()) != null){

            if(line.startsWith(APData.FBUS31)){
```

```
                readThisData = true;
            } else if(line.startsWith(APData.FBUS)){
                readThisData = false;
            }

            if(!readThisData || line.length() < 5) continue;

            if(line.startsWith(APData.VALID)){
                startTime = Integer.parseInt(line.substring(24,28));
            }

            String station = line.substring(1,4);

            // Read data by flight level
            if(wind_.containsWindStation(station)){
                for(int i = 0; i < Wind.flightLevel.length; i++){
                    String windData = line.substring(startIndex[i], end...
...Index[i]);
                    wind_.addWindData(station, windData, startTime, Win...
...d.flightLevel[i]);
                }
            }

        }
        line = null;

        return this;
    }

    public FAPioWindReader readClosestWindStations() throws IOException...
...{
        String line;
        BufferedReader d;
        for(int i=0; i<3; i++){
            if(i == 0){
                d = new BufferedReader(new FileReader(Settings.windNava...
...idClosestStation));
            }else if(i == 1){
                d = new BufferedReader(new FileReader(Settings.windFixC...
...losestStation));
            }else{   // i == 2
                d = new BufferedReader(new FileReader(Settings.windNatf...
...ixClosestStation));
            }

            while((line = d.readLine()) != null){
                //String[] dataFields = line.split(APData.COMMA);
```

```
                //if(dataFields.length < 2) continue;

                int firstCommaIdx = line.indexOf(APData.COMMA);
                if(firstCommaIdx < 0) continue;

                // identifier, wind station
                wind_.addClosestStationToIdentifier(
                        line.substring(0,firstCommaIdx),
                        line.substring(firstCommaIdx+1, line.length()))...
...;
                //dataFields[0],
                // dataFields[1]);
            }
            d.close();

        }

        line = null;

        return this;
    }

}
```

FAPioWindWriter.java

```
/*
 * FAPioWindWriter.java
 *
 * Created on May 26, 2007, 4:20 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author Jeff
 */
import java.util.*;
import java.io.*;
public class FAPioWindWriter {
    private final int WIND_FILE_CHARACTERS = 50000;

    Wind wind_;
    String windDirectory_;


    /** Creates a new instance of FAPioWindWriter */
    public FAPioWindWriter(Wind wind, String windDirectory) {
        wind_ = wind;
        windDirectory_ = windDirectory;
    }

    public FAPioWindWriter writeWindData() throws IOException{
        //final int[] flightLevel = {30, 60, 90, 120, 180, 240, 300, 34...
...0, 390};

        // Index j is for each time interval 0,600,1800,2400
        for(int j = 0; j < 4; j++){
            // Write the data
            String windDataOutFile = windDirectory_ + Integer.toString(...
...j*600) + "ZWindData.csv";
            PrintWriter outWind = new PrintWriter(new BufferedWriter(ne...
...w FileWriter(windDataOutFile)));

            StringBuilder sWind = new StringBuilder(WIND_FILE_CHARACTER...
...S);

            Iterator<String> iterator = wind_.getWindStationIterator();...
...
            while(iterator.hasNext()){
```

```
                String station = iterator.next();

                for(int i = 0; i < Wind.flightLevel.length; i++){
                    sWind.append(wind_.getWindStationLatitude(station))...
...; sWind.append(APData.COMMA);
                    sWind.append(wind_.getWindStationLongitude(station)...
...); sWind.append(APData.COMMA);
                    sWind.append(Wind.flightLevel[i]); sWind.append(APD...
...ata.COMMA);
                    sWind.append(wind_.getWindDirection(station, j, i))...
...; sWind.append(APData.COMMA);
                    sWind.append(wind_.getWindSpeed(station, j, i)); sW...
...ind.append(APData.SLASH_N);
                    //sWind.append(station); sWind.append(",");
                    //sWind.append(a.getWindString(station, j, i)); sWi...
...nd.append("\n");
                }
            }

            outWind.print(sWind.toString());
            outWind.close();

            sWind = null;

        }

        return this;
    }

    public FAPioWindWriter writeWindStations() throws IOException{
        String stationOutFile = Settings.windStations.replace(".txt","....
...csv");
        PrintWriter out = new PrintWriter(new BufferedWriter(new FileWr...
...iter(stationOutFile)));

        StringBuilder s = new StringBuilder();

        Iterator<String> e = wind_.getWindStationIterator();

        while(e.hasNext()){
            String station = e.next();

            s.append(station); s.append(APData.COMMA);
            s.append(wind_.getWindStationLatitude(station)); s.append(A...
...PData.COMMA);
            s.append(wind_.getWindStationLongitude(station)); s.append(...
...APData.SLASH_N);
```

```
        }

        out.print(s.toString());
        out.close();

        s = null;

        return this;
    }


}
```

Flight.java

```
//
//  Flight.java
//  AirPlanJ
//
//  Created by jeff on 8/4/06.
//  Copyright 2006. All rights reserved.
//
import java.util.*;
public class Flight implements Comparable<Flight> {
    private static final double HOURSTOMINUTES = 60.0D;
    private static final double SECONDSTOMINUTES = (1/60.0D);

    private String callSign_;
    private Airport originAirport_;
    private Airport destinationAirport_;
    private EAircraft aircraftType_;
    private int cruiseFlightLevel_;
    private FTrajectory plannedTrajectory_;
    private FTrajectory radarTrackTrajectory_;
    private GregorianCalendar departureTime_;


    public Flight(String callSign, Airport originAirport, Airport desti...
...nationAirport,
            EAircraft aircraftType, int cruiseFlightLevel){
        callSign_ = callSign;
        originAirport_ = originAirport;
        destinationAirport_ = destinationAirport;
        aircraftType_ = aircraftType;
        cruiseFlightLevel_ = cruiseFlightLevel;
    }

    public Flight(String callSign, EAircraft aircraftType){
        callSign_ = callSign;
        aircraftType_ = aircraftType;
        cruiseFlightLevel_ = -1;
    }

    public Flight(String callSign){
        callSign_ = callSign;
        aircraftType_ = new EAircraft(APData.NULLSTRING);
    }

    /** Set the departure time */
    public Flight setDepartureTime(GregorianCalendar departureTime){
        departureTime_ = departureTime;
```

```
    return this;
}

public double getDepartureTimeMinutes(){
    int hour = departureTime_.get(Calendar.HOUR_OF_DAY);
    int minute = departureTime_.get(Calendar.MINUTE);
    int second = departureTime_.get(Calendar.SECOND);

    return hour*HOURSTOMINUTES + minute + second*SECONDSTOMINUTES;
}

public int compareTo(Flight f){
    return Double.compare(this.getDepartureTimeMinutes(),
            f.getDepartureTimeMinutes());
}

public String getDepartureTimeString(){
    StringBuilder depTime = new StringBuilder();

    int hour = departureTime_.get(Calendar.HOUR_OF_DAY);
    int minute = departureTime_.get(Calendar.MINUTE);
    int second = departureTime_.get(Calendar.SECOND);

    if(hour < 10) depTime.append(0);
    depTime.append(hour);

    if(minute < 10) depTime.append(0);
    depTime.append(minute);

    if(second < 10) depTime.append(0);
    depTime.append(second);

    return depTime.toString();
}

/** Flight identifier */
public String getCallSign(){
    return callSign_;
}

/** Origin airport of type Airport for flight */
public Airport getOriginAirport(){
    return originAirport_;
}

/** Destination airport of type Airport for flight */
```

```
public Airport getDestinationAirport(){
    return destinationAirport_;
}

/** Aircraft type of type EAircraft for flight */
public EAircraft getAircraftType(){
    return aircraftType_;
}
public Flight addAircraftType(EAircraft ac){
    aircraftType_ = ac;
    return this;
}

/** Planned cruising flight level for flight */
public int getCruisingFlightLevel(){
    return cruiseFlightLevel_;
}

public Flight setCruisingFlightLevel(int cruiseFlightLevel){
    cruiseFlightLevel_ = cruiseFlightLevel;
    return this;
}

/** Set the planned trajectory for the flight */
public Flight setPlannedTrajectory(FTrajectory t){
    plannedTrajectory_ = t;
    return this;
}

/** Check if a planned trajectory has been set for this flight */
public boolean hasPlannedTrajectory(){
    return !(plannedTrajectory_ == null);
}

/** Planned trajectory of type FTrajectory */
public FTrajectory getPlannedTrajectory(){
    return plannedTrajectory_;
}

/** Set the radar track trajectory for the flight */
public Flight setRadarTrackTrajectory(FTrajectory t){
    radarTrackTrajectory_ = t;
    return this;
}

/** Radar tracks of type FTrajectory */
public FTrajectory getRadarTrackTrajectory(){
```

```
        try{
            int l = radarTrackTrajectory_.noWaypoints();
        }catch(RuntimeException e){
            radarTrackTrajectory_ = new FTrajectory();
        }

        return radarTrackTrajectory_;
    }
}
```

FlightTraversal.java

```
/*
 * FlightTraversal.java
 *
 * Created on September 3, 2007, 12:04 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author jeff
 */
import java.util.*;
public class FlightTraversal {
    public String FlightId;

    public String Airport;

    public ArrayList<String> PlannedSector;
    public ArrayList<Double> PlannedEntryMin;
    public ArrayList<Double> PlannedExitMin;
    public ArrayList<Double> PlannedTraversalTimeMin;

    public ArrayList<String> ObservedSector;
    public ArrayList<Double> ObservedEntryMin;
    public ArrayList<Double> ObservedExitMin;
    public ArrayList<Double> ObservedTraversalTimeMin;

    /* Arraylist to store:
        - planned traversal time
     *  - observed traversal time
     *  - observed/planned ratio
     *  - sector 2D density
     *  - sector 3D density
     */
    public ArrayList<ArrayList<Double>> ratioValues;

    /** Creates a new instance of FlightTraversal */
    public FlightTraversal(String flightId) {
        FlightId = flightId;

        PlannedSector = new ArrayList<String>();
        PlannedEntryMin = new ArrayList<Double>();
        PlannedExitMin = new ArrayList<Double>();
        PlannedTraversalTimeMin = new ArrayList<Double>();
```

```
        ObservedSector = new ArrayList<String>();
        ObservedEntryMin = new ArrayList<Double>();
        ObservedExitMin = new ArrayList<Double>();
        ObservedTraversalTimeMin = new ArrayList<Double>();
    }

    public double hitRate (){
        double totalCount = 0.0D;
        double hits = 0.0D;
        for(String sec1:PlannedSector){

            for(String sec2:ObservedSector){
                totalCount += 1.0D;
                if(sec1.equals(sec2)) hits += 1.0D;
            }
        }

        if(totalCount == 0.0D){
            return 0.0D;
        }else if(hits > PlannedSector.size()){
            return 1.0D;
        }else{
            return hits/PlannedSector.size();
        }
    }

    public double weightedHitRate(){
        double totalObservedTime = 0.0D;
        double hitTime = 0.0D;

        for(double tt:ObservedTraversalTimeMin) totalObservedTime += tt...
...;

        for(int i=0; i<PlannedSector.size(); i++){
            String sec1 = PlannedSector.get(i);
            for(String sec2:ObservedSector){
                if(sec1.equals(sec2)){
                    hitTime += PlannedTraversalTimeMin.get(i);
                }
            }
        }

        if(totalObservedTime == 0.0D){
            return 0.0D;
        }
        return hitTime/totalObservedTime;
```

```
    }

    public double totalPlannedTime(){
        double totalPlannedTime = 0.0D;

        for(double tt:PlannedTraversalTimeMin) totalPlannedTime += tt;

        return totalPlannedTime;
    }

    public double averageDensity(HashMap<String, SectorTraversal> secto...
...rTraversals,
            boolean use3d){
        double totalCount = 0.0D;
        double totalDensity = 0.0D;

        for(int i=0; i<PlannedSector.size(); i++){
            String sectorS = PlannedSector.get(i);
            if(sectorTraversals.containsKey(sectorS)){
                SectorTraversal sector = sectorTraversals.get(sectorS);...
...
                int plannedEntry = (int)Math.round(PlannedEntryMin.get(...
...i));
                int plannedExit = (int)Math.round(PlannedExitMin.get(i)...
...);
                if(plannedExit > 1439) plannedExit = 1439;
                for(int j=plannedEntry; j<=plannedExit; j++){
                    double density = use3d ?
                          sector.ObservedDensityByMin3D.get(j)
                        : sector.ObservedDensityByMin2D.get(j);

                    totalCount += 1.0D;
                    totalDensity += density;
                }
            }
        }

        if(totalCount == 0.0D){
            return 0.0D;
        }else{
            return totalDensity/totalCount;
        }
    }

    public FlightTraversal calculateObservedPlannedRatio(HashMap<String...
...,SectorTraversal> sectorTraversals){
```

```
        ratioValues = new ArrayList<ArrayList<Double>>();

        for(int i=0; i<PlannedSector.size(); i++){
            String plannedSectorS = PlannedSector.get(i);
            double plannedEntryMin = PlannedEntryMin.get(i);
            double plannedExitMin = PlannedExitMin.get(i);
            double error = Double.MAX_VALUE;
            int observedSectorIdx = -1;

            for(int j=0; j<ObservedSector.size(); j++){
                String observedSectorS = ObservedSector.get(j);
                if(!observedSectorS.equals(plannedSectorS)) continue;

                double observedEntryMin = ObservedEntryMin.get(j);
                double errorObs = Math.abs(plannedEntryMin - observedEn...
...tryMin);
                if(!(errorObs < error)) continue;
                error = errorObs;
                observedSectorIdx = j;
            }

            if(observedSectorIdx == -1) continue;

            SectorTraversal sectorTraversal = sectorTraversals.get(plan...
...nedSectorS);

            for(long k=Math.round(plannedEntryMin); k<=Math.round(plann...
...edExitMin); k++){
                int k2 = (int)k;

                /* Only consider observations with non-zero sector trav...
...ersal times. */
                if(!(PlannedTraversalTimeMin.get(i) > 0)) continue;

                ArrayList<Double> ratioData = new ArrayList<Double>(5);...
...
                ratioData.add(PlannedTraversalTimeMin.get(i));
                ratioData.add(ObservedTraversalTimeMin.get(observedSect...
...orIdx));
                ratioData.add(ObservedTraversalTimeMin.get(observedSect...
...orIdx)/
                    PlannedTraversalTimeMin.get(i));

                ratioData.add(sectorTraversal.ObservedDensityByMin2D.ge...
...t(k2));
                ratioData.add(sectorTraversal.ObservedDensityByMin3D.ge...
...t(k2));
```

```
                ratioValues.add(ratioData);
            }
        }

        return this;
    }

}
```

FTrajectory.java

```
//
//  FTrajectory.java
//  AirPlanJ
//
//  Created by jeff on 8/4/06.
//  Copyright 2006. All rights reserved.
//
import java.util.*;
public class FTrajectory {
    /** Initial capacity for number of waypoints that defines a traject...
...ory */
    static final int INIT_NUM_WAYPOINTS = 50;

    private GregorianCalendar departureTime_;
    private ArrayList<GPoint4D <Double>> waypoints_;
    private ArrayList<String> waypointIdentifiers_;
    private ArrayList<Boolean> isClimbing_;
    private ArrayList<Double> groundSpeed_;

    /** Conversion of milliseconds to minutes */
    static final double millisToMinutes = 1D/(60D*1000D);

    /** Trajectory with unknown departure time */
    public FTrajectory(){
        waypoints_ = new ArrayList <GPoint4D <Double>>(INIT_NUM_WAYPOIN...
...TS);
        waypointIdentifiers_ = new ArrayList <String>(INIT_NUM_WAYPOINT...
...S);
        isClimbing_ = new ArrayList <Boolean>(INIT_NUM_WAYPOINTS);
        groundSpeed_ = new ArrayList <Double>(INIT_NUM_WAYPOINTS);
    }

    /** Trajectory with known deparure time */
    public FTrajectory(GregorianCalendar departureTime){
        departureTime_ = departureTime;
        waypoints_ = new ArrayList <GPoint4D <Double>>(INIT_NUM_WAYPOIN...
...TS);
        waypointIdentifiers_ = new ArrayList <String>(INIT_NUM_WAYPOINT...
...S);
        isClimbing_ = new ArrayList <Boolean>(INIT_NUM_WAYPOINTS);
        groundSpeed_ = new ArrayList <Double>(INIT_NUM_WAYPOINTS);
    }

    /** Check for invalid flight levels that end with '0' */
    public FTrajectory checkFlightLevelPersistence(){
        for(int i=0; i < waypoints_.size(); i++){
```

```
            GPoint4D<Double> waypoint = waypoints_.get(i);

            // Only apply filter to flights that end with a '0'
            if(waypoint.getElevation()%10 > 0.01D){
                continue;
            }

            int sequentialValues = 1;
            // Check previous 3 values
            for(int j=1; j < 4; j++){
                if(i-j < 0) break;
                GPoint4D<Double> waypoint2 = waypoints_.get(i-j);
                if(Math.abs(waypoint.getElevation() - waypoint2.getElev...
...ation()) < 0.01D){
                    sequentialValues++;
                }
                else break;
            }

            // Check next 3 values
            for(int j=1; j < 4; j++){
                if(i+j >= waypoints_.size()) break;
                GPoint4D<Double> waypoint2 = waypoints_.get(i+j);
                if(Math.abs(waypoint.getElevation() - waypoint2.getElev...
...ation()) < 0.01D){
                    sequentialValues++;
                }
                else break;
            }

            if(sequentialValues < 4) waypoint.setElevation(-1.0D);
        }

        return this;
    }

    /** Get the number of waypoints in an altitude range. */
    public int numberOfWaypointsInAltitudeRange(double minAltitude, dou...
...ble maxAltitude){
        int numInRange = 0;
        for(int i=0; i< waypoints_.size(); i++){
            GPoint4D<Double> waypoint = waypoints_.get(i);

            if(waypoint.getElevation() >= minAltitude && waypoint.getEl...
...evation() <= maxAltitude){
                numInRange++;
            }
```

```
        }
        return numInRange;
    }

    // Creates the 4D Trajectory
    public FTrajectory build4DTrajectoryFromWaypoints(ArrayList<String>...
... elementIds,
            ArrayList<GPoint2D<Double>> locations, EAircraft aircraftTy...
...pe){

        // Set identifiers
        waypointIdentifiers_ = elementIds;

        // Clear existing waypoint locations (if any)
        waypoints_.clear();

        for(int i=0; i<locations.size(); i++){
            GPoint2D<Double> pt2 = locations.get(i);
            GPoint4D<Double> pt4 = new GPoint4D<Double>(pt2.getLongitud...
...e(), pt2.getLatitude());

            waypoints_.add(pt4);
        }

        return this;
    }


    /** Add a waypoint to the trajectory */
    public FTrajectory addWaypoint(GPoint4D<Double> waypoint, EAircraft...
... aircraft){
        if(waypoints_.size() > 0){
            GPoint4D<Double> prevWaypoint = waypoints_.get(waypoints_.s...
...ize()-1);

            // Check if point is not duplicate
            if(waypoint.getTime().getTimeInMillis() <= prevWaypoint.get...
...Time().getTimeInMillis()){
                return this;
            }

            //waypoints_.add(waypoint);

            int i=1;
            do{
                prevWaypoint = waypoints_.get(waypoints_.size() - i);
```

```
                if(prevWaypoint.getElevation() > 0.0){
                    // Check that rate of climb/descent does not exceed...
... aircraft performance
                    Long timeDiff = waypoint.getTime().getTimeInMillis(...
...) - prevWaypoint.getTime().getTimeInMillis();
                    double timeDiffMinutes = timeDiff.doubleValue()*mil...
...lisToMinutes;

                    double altitudeDiff = waypoint.getElevation() - pre...
...vWaypoint.getElevation();
                    double rateOfAltitudeChange = Math.abs(100*altitude...
...Diff/timeDiffMinutes);
                    double maxRateOfAltitudeChange = 0.0;

                    if(altitudeDiff < 0.0){
                        // Descending
                        maxRateOfAltitudeChange = aircraft.getCallSign(...
...).length() == 0 ?
                            0.95D*EAircraft.getMaxRateOfDescent(prevWay...
...point.getElevation()) :
                            0.95D*aircraft.getRateOfDescent(prevWaypoin...
...t.getElevation());
                    }
                    else{
                        // Climbing
                        maxRateOfAltitudeChange = aircraft.getCallSign(...
...).length() == 0 ?
                            1.05D*EAircraft.getMaxRateOfClimb(prevWaypo...
...int.getElevation()) :
                            1.05D*aircraft.getRateOfClimb(prevWaypoint....
...getElevation());
                    }

                    // Altitude not valid if change exceeds max rate by...
... 10%
                    if(rateOfAltitudeChange > maxRateOfAltitudeChange){...
...                         waypoint.setElevation(-1.0);    // Set to inval...
...id altitude
                    }
                    waypoints_.add(waypoint);
                    return this;
                }
                i++;
            }while(i<waypoints_.size());

        }
        else{
```

```
        waypoints_.add(waypoint);
    }
    return this;
}

/** Add climbing status to the trajectory */
public FTrajectory addClimbingStatus(Boolean status){
    isClimbing_.add(status);
    return this;
}

/** Number of waypoints used to iterate through waypoint list
 * using getWaypoint method */
public int noWaypoints(){
    return waypoints_.size();
}

/** Get a trajectory waypoint by index */
public GPoint4D<Double> getWaypoint(int index){
    return waypoints_.get(index);
}

/** Get identifier for the waypoint */
public String getWaypointIdentifier(int index){
    return waypointIdentifiers_.get(index);
}

/** Returns the last waypoint added to the trajectory */
public GPoint4D<Double> getLastWaypoint(){
    return waypoints_.get(waypoints_.size()-1);
}

/** Trim waypoints to reduce storage space */
public FTrajectory trimWaypoints(){
    waypoints_.trimToSize();
    return this;
}

/** Get climbing status by index */
public Boolean getClimbingStatus(int index){
    return isClimbing_.get(index);
}

/** Trim climbing status to reduce storage space */
public FTrajectory trimClimbingStatus(){
    isClimbing_.trimToSize();
    return this;
```

```
    }

    /** Add ground speed to the trajectory */
    public FTrajectory addGroundSpeed(String groundSpeedS){
        double groundSpeedD = -1.0;

        if(groundSpeedS.length() > 0){
            groundSpeedD = Double.parseDouble(groundSpeedS);
        }

        // Check for unrealistic groundspeeds
        groundSpeedD = groundSpeedD > 900.0 ? -1.0 : groundSpeedD;

        // Add the ground speed
        groundSpeed_.add(groundSpeedD);

        return this;
    }

    /** Get ground speed by index */
    public Double getGroundSpeed(int index){
        return groundSpeed_.get(index);
    }

    public FTrajectory trimArrayLists(){
        waypoints_.trimToSize();
        waypointIdentifiers_.trimToSize();
        isClimbing_.trimToSize();
        groundSpeed_.trimToSize();

        return this;
    }
}
```

GPoint2D.java

```
/*
 * GPoint2D.java
 *
 * Created on July 24, 2006, 8:27 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
/**
 *
 * @author jeff
 */
public class GPoint2D<E extends Number>{
    private E longitude_;
    private E latitude_;

    /** Creates a new instance of GPoint2D
     * a geographic (longitude,latitude) pair
     */
    public GPoint2D() {

    }

    /** Creates new instance of GPoint2D
     * (longitude, latitude) pair of type E
     */
    public GPoint2D(E longitude, E latitude){
        longitude_ = longitude;
        latitude_ = latitude;
    }

        /** Get longitude */
    public E getLongitude(){
        return longitude_;
    }

    /** Set longitude */
    public GPoint2D setLongitude(E longitude){
        longitude_ = longitude;
        return this;
    }

    /** Get latitude */
    public E getLatitude(){
        return latitude_;
    }
```

```
    /** Set latitude */
    public GPoint2D setLatitude(E latitude){
        latitude_ = latitude;
        return this;
    }


}
```

GPoint3D.java

```
/*
 * GPoint3D.java
 *
 * Created on July 24, 2006, 9:26 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
/**
 *
 * @author jeff
 */
public class GPoint3D<E extends Number> extends GPoint2D<E> {
    private E elevation_;

    /** Creates a new instance of GPoint3D
     * a geographic (longitude,latitude,elevation
     * triple
     */
    public GPoint3D() {
    }

    /** Creates new instance of GPoint3D
     * (longitude, latitude,elevation) triple
     * of type E
     */
    public GPoint3D(E longitude, E latitude, E elevation){
        super.setLongitude(longitude);
        super.setLatitude(latitude);
        elevation_ = elevation;
    }

    public GPoint3D(GPoint3D<E> pt){
        super.setLongitude(pt.getLongitude());
        super.setLatitude(pt.getLatitude());
        elevation_ = pt.getElevation();
    }

    /** Get elevation */
    public E getElevation(){
        return elevation_;
    }

    /** Set elevation */
    public GPoint3D setElevation(E elevation){
        elevation_ = elevation;
```

```
        return this;
    }

}
```

GPoint4D.java

```
/*
 * GPoint4D.java
 *
 * Created on August 4, 2006, 10:19 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
/**
 *
 * @author jeff
 */

 import java.util.*;

public class GPoint4D<E extends Number> extends GPoint3D<E> {
    private GregorianCalendar time_;

    /** Creates a new instance of GPoint4D
     * a geographic (longitude,latitude,elevation,time)
     */
    public GPoint4D() {
    }

    /** Creates new instance of GPoint4D
     * (longitude, latitude,elevation,time)
     * of type E
     */
    public GPoint4D(E longitude, E latitude, E elevation, GregorianCale...
...ndar time){
        super.setLongitude(longitude);
        super.setLatitude(latitude);
        super.setElevation(elevation);
        time_ = time;
    }

    public GPoint4D(E longitude, E latitude){
         super.setLongitude(longitude);
         super.setLatitude(latitude);
    }

    /** Get elevation */
    public GregorianCalendar getTime(){
        return time_;
    }
```

```
    /** Set elevation */
    public GPoint4D setTime(GregorianCalendar time){
        time_ = time;
        return this;
    }

}
```

KernelDensity.java

```
/*
 * KernelDensity.java
 *
 * Created on September 7, 2007, 2:44 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author jeff
 */
import java.util.*;
public class KernelDensity {

    private String description_;

    private ArrayList<Double> values_;
    private ArrayList<Double> densities_;

    /** Creates a new instance of KernelDensity */
    public KernelDensity(String description) {
        description_ = description;

        values_ = new ArrayList<Double>();
        densities_ = new ArrayList<Double>();
    }

    public KernelDensity(String description, ArrayList<Double> values,
            ArrayList<Double> densities){
        description_ = description;

        values_ = values;
        densities_ = densities;
    }

    public String getDescription(){
        return description_;
    }

    public KernelDensity addPoint(double value, double density){
        values_.add(value);
        densities_.add(density);

        return this;
```

```
    }

    public double getValue(int index){
        return values_.get(index);
    }

    public double getDensity(int index){
        return densities_.get(index);
    }

    public KernelDensity setDensity(int index, double density){
        densities_.set(index, density);

        return this;
    }

    public int noPoints(){
        return values_.size();
    }

    public ArrayList<Double> getValues(){
        return values_;
    }

    public ArrayList<Double> getDensities(){
        return densities_;
    }

    public Double[] densityArray(){
        Double[] tmpDouble = new Double[densities_.size()];
        return densities_.toArray(tmpDouble);
        //return densities_.toArray();
    }

//    public KernelDensity removeNonIntegerAndNormalize(){
//        if(description_.startsWith("ratios")) return this;
//
//        ArrayList<Double> newValues = new ArrayList<Double>(values_.s...
...ize());
//        ArrayList<Double> newDensities = new ArrayList<Double>(densit...
...ies_.size());
//
//        // Total density for normalization
//        double totalDensity = 0.0D;
//
//        for(int i=0; i<values_.size(); i++){
//            if(i % 2 != 0) continue; // Skip odd values
```

```
//
//            newValues.add(values_.get(i));
//            newDensities.add(densities_.get(i));
//
//            totalDensity += densities_.get(i);
//        }
//
//        // Perform normalization
//        for(int i=0; i<newDensities.size(); i++){
//            double currentDensity = newDensities.get(i);
//            newDensities.set(i, currentDensity/totalDensity);
//        }
//
//        values_ = newValues;
//        densities_ = newDensities;
//
//        return this;
//    }


}
```

KernelReader.java

```
/*
 * KernelReader.java
 *
 * Created on September 7, 2007, 5:23 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author jeff
 */
import java.util.*;
import java.io.*;
public class KernelReader {

    /** Creates a new instance of KernelReader */
    public KernelReader() {
    }

    public KernelReader readKernelDensities(String fileName) throws IOE...
...xception{
        HashMap<String, KernelDensity> densities = APData.kernelDensiti...
...es;

        BufferedReader d = new BufferedReader(new FileReader(fileName))...
...;
        String line;

        while((line = d.readLine()) != null){
            //String[] dataFields = line.split(",");
            //if(dataFields.length < 4) continue;

            int idx1 = line.indexOf(APData.COMMA);

            if(idx1 < 0) continue;

            int idx2 = line.indexOf(APData.COMMA, idx1+1);
            int idx3 = line.indexOf(APData.COMMA, idx2+1);

            //String name = dataFields[0].trim();
            String name = line.substring(0, idx1);

            // Only consider look-ahead times of 0 minutes for now
            int lat = Integer.parseInt(line.substring(idx1+1,idx2));//d...
```

```
...ataFields[1].trim());
            if(lat > 0) continue;

            double value = Double.parseDouble(line.substring(idx2+1,idx...
...3));//dataFields[2].trim());
            double density = Double.parseDouble(line.substring(idx3+1,l...
...ine.length()));//dataFields[3].trim());

            KernelDensity k = densities.containsKey(name) ?
                densities.get(name) : new KernelDensity(name);

            densities.put(name, k);

            k.addPoint(value, density);
        }

        d.close();

        line = null;
        return this;
    }

    // Change all distributions but ratio distributions to integer x va...
...lues
//    public KernelReader writeRevisedKernelDensities(String fileName) ...
...throws IOException{
//        HashMap<String, KernelDensity> densities = APData.kernelDensi...
...ties;
//
//        PrintWriter out = new PrintWriter(new BufferedWriter(new File...
...Writer(fileName)));
//
//        StringBuilder s = new StringBuilder();
//
//        for(KernelDensity kernelDensity:densities.values()){
//            kernelDensity.removeNonIntegerAndNormalize();
//
//            ArrayList<Double> densitiesOut = kernelDensity.getDensiti...
...es();
//            ArrayList<Double> valuesOut = kernelDensity.getValues();
//
//            for(int i=0; i<densitiesOut.size(); i++){
//                s.append(kernelDensity.getDescription());
//                s.append(",0,");
//
//                // Cast value as an integer for non-ratio distributio...
...ns
```

```
//                if(kernelDensity.getDescription().startsWith("ratios"...
...)){
//                    s.append(valuesOut.get(i));
//                }else{
//                    s.append(valuesOut.get(i).intValue());
//                }
//
//                s.append(",");
//
//                // Write density values
//                s.append(densitiesOut.get(i));
//                s.append("\n");
//            }
//
//        }
//
//        out.print(s.toString());
//        out.close();
//
//
//        return this;
//    }

}
```

LinearInterpolator.java

```
/*
 * LinearInterpolator.java
 *
 * Created on October 23, 2007, 12:15 AM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author Jeff
 */
import java.util.*;
public class LinearInterpolator {
    private ArrayList<Double> x_;
    private ArrayList<Double> y_;
    private double binWidth_;
    private int n_;

    /** Creates a new instance of LinearInterpolator */
    public LinearInterpolator(ArrayList<Double> x, ArrayList<Double> y)...
... {
        x_ = x;
        y_ = y;

        // Calculate the bin width for vector x
        n_ = x_.size();
        binWidth_ = (x_.get(n_-1) - x_.get(0)) / (n_-1);
    }

    public double interpolateAtValue(double xi){
        // Return 0 if out of range
        if(xi < x_.get(0) || xi > x_.get(n_-1)) return 0D;

        // Find the bin number
        int k = 0;
        for(int i=0; i < x_.size()-1; i++){
            double x_lower = x_.get(i);
            double x_upper = x_.get(i+1);
            if(xi >= x_lower && xi <= x_upper){
                k = i;
                break;
            }
        }
```

```
        // Perform interpolation
        double s = (xi - x_.get(k)) / binWidth_;
        double yi = y_.get(k) + s * (y_.get(k+1) - y_.get(k));

        return yi;
    }

}
```

Link3D.java

```
/*
 * Link3D.java
 *
 * Created on May 25, 2007, 2:04 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author jeff
 */
import java.util.*;
public class Link3D{
    // Define what type of link this is.
    public enum LinkTypes {
        Intrasector, AirportConnector, JetRoute, VictorAirway, RnavRout...
...e,
            AdditionalLinks, Copy
    }

    private String identifier_;

    private Node3D<Double> fromNode_;
    private String fromNodeId_;

    private Node3D<Double> toNode_;
    private String toNodeId_;

    private double length_;
    private double azimuth_;

    // Parent sector (where this link is located)
    private ASector parentSector_;

    // Link type for display purposes
    private final LinkTypes linkType_;

    // Altitude limits for link
    private double floorAltitude_;
    private double ceilingAltitude_;

    private double[] doubleValues_;

    /** Creates a new instance of Link3D */
```

```
    public Link3D(String identifier, Node3D<Double> fromNode, String fr...
...omNodeId,
            Node3D<Double> toNode, String toNodeId, LinkTypes linkType)...
...{

        identifier_ = identifier;
        fromNode_ = fromNode;
        fromNodeId_ = fromNodeId;
        toNode_ = toNode;
        toNodeId_ = toNodeId;
        length_ = -1.0D;
        linkType_ = linkType;
        floorAltitude_ = -999.0D;
        ceilingAltitude_ = -999.0D;
    }

    public Link3D copy(){
        return new Link3D(identifier_, fromNode_, fromNodeId_, toNode_,...
... toNodeId_, linkType_);
    }

    public Link3D initializeDoubleValues(int numValues){
        doubleValues_ = new double[numValues];
        for(int i=0; i<doubleValues_.length; i++) doubleValues_[i] = 0....
...0D;
        return this;
    }

    public Link3D setDoubleValue(int index, double val){
        doubleValues_[index] = val;
        return this;
    }

    public double getDoubleValue(int index){
        return doubleValues_[index];
    }

    public String getIdentifier(){
        return identifier_;
    }

    public Node3D<Double> getFromNode(){
        return fromNode_;
    }

    public String getFromNodeId(){
        return fromNodeId_;
```

```
}

public Node3D<Double> getToNode(){
    return toNode_;
}

public String getToNodeId(){
    return toNodeId_;
}

public double getLength(){
    // Set length if too short
    if(length_ < 0.001D){
        Mapping m = new Mapping(fromNode_.getLatitude(),
                fromNode_.getLongitude(),
                toNode_.getLatitude(),
                toNode_.getLongitude());

        double len = m.getDistance();

        length_ = Math.max(len, 0.001D);
    }

    return length_;
}

public Link3D setLength(double length){
    if(length == 0.0D) length = 0.001D;
    length_ = length;
    return this;
}

public double getAzimuth(){
    return azimuth_;
}

public Link3D setAzimuth(double azimuth){
    azimuth_ = azimuth;
    return this;
}

public boolean lengthWasSet(){
    return (length_ >= 0.0D);
}

public Link3D setParentSector(ASector sector){
    parentSector_ = sector;
```

```
        return this;
    }

    public ASector getParentSector(){
        return parentSector_;
    }

    public LinkTypes getLinkType(){
        return linkType_;
    }

    public Link3D setAltitudeLimits(double floorAltitude, double ceilin...
...gAltitude){
        floorAltitude_ = Math.min(floorAltitude, ceilingAltitude);
        ceilingAltitude_ = Math.max(floorAltitude, ceilingAltitude);
        return this;
    }

    public double getFloorAltitude(){
        return Math.min(floorAltitude_, ceilingAltitude_);
    }

    public double getCeilingAltitude(){
        return Math.max(floorAltitude_, ceilingAltitude_);
    }

    public boolean isAirportConnector(){
        return linkType_.equals(LinkTypes.AirportConnector);
    }

    public boolean isAirway(){
        return linkType_.equals(LinkTypes.JetRoute) ||
                linkType_.equals(LinkTypes.RnavRoute) ||
                linkType_.equals(LinkTypes.VictorAirway);

    }

    /** Wind speed along link [knots] */
    public double getWindSpeed(double timeMinutes, double flightLevel){...
...        String fromId = fromNode_.getAirspaceIdentifier();
        String toId = toNode_.getAirspaceIdentifier();

        double fromWindSpeed = APData.wind.getWindSpeedByAirspaceIdenti...
...fier(
                fromId, timeMinutes, flightLevel);

        double toWindSpeed = APData.wind.getWindSpeedByAirspaceIdentifi...
```

```
...er(
                toId, timeMinutes, flightLevel);

        return 0.5D * (fromWindSpeed + toWindSpeed);
    }

    /** Wind direction along link [degrees] */
    public double getWindDirection(double timeMinutes, double flightLev...
...el){
        String fromId = fromNode_.getAirspaceIdentifier();
        String toId = toNode_.getAirspaceIdentifier();

        double fromWindDirection = APData.wind.getWindDirectionByAirspa...
...ceIdentifier(
                fromId, timeMinutes, flightLevel);

        double toWindDirection = APData.wind.getWindDirectionByAirspace...
...Identifier(
                toId, timeMinutes, flightLevel);

        return 0.5D * (fromWindDirection + toWindDirection);
    }

    public boolean leadsTo(Node3D<Double> n) {
        return (toNode_.equals(n));
    }

    public Node3D<Double> getConnected(Node3D<Double> n) {
        if (fromNode_.equals(n)) {
            return toNode_;
        } else {
            return null;
        }
    }




}
```

Mapping.java

```
/*
 * Mapping.java
 *
 * Created on October 18, 2007, 3:13 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author jeff
 */
public class Mapping {
    // Earth geometry [nm]
    private static final double SEMI_MAJOR_AXIS = 3443.9D;
    private static final double SEMI_MINOR_AXIS = 3432.4D;
    private static final double ECCENTRICITY    = 0.0818D;

    // Latitude/longitude coordinates [degrees]
    private double lat1_;
    private double lon1_;
    private double lat2_;
    private double lon2_;

    // Distance [nm]/azimuth [degrees] between latitude and longitude i...
...n degrees
    private double distance_;
    private double azimuth_;

    /** Creates a new instance of Mapping */
    public Mapping(double lat1, double lon1, double lat2, double lon2) ...
...{
        lat1_ = lat1;
        lon1_ = lon1;
        lat2_ = lat2;
        lon2_ = lon2;
    }

    private Mapping calculateDistance(){
        // Code converted from MATLAB mapping toolbox distance function...
...
        // Convert latitude/longitude to radians
        double lat1r = Math.toRadians(lat1_);
        double lon1r = Math.toRadians(lon1_);
        double lat2r = Math.toRadians(lat2_);
```

```
        double lon2r = Math.toRadians(lon2_);

        // Compute parametric latitudes
        double par1 = Math.atan((SEMI_MINOR_AXIS/SEMI_MAJOR_AXIS)*
                Math.tan(lat1r));
        double par2 = Math.atan((SEMI_MINOR_AXIS/SEMI_MAJOR_AXIS)*
                Math.tan(lat2r));

        // Convert spherical coordinates to cartesian coordinates
        double x1 = SEMI_MAJOR_AXIS*Math.cos(par1)*Math.cos(lon1r);
        double y1 = SEMI_MAJOR_AXIS*Math.cos(par1)*Math.sin(lon1r);
        double z1 = SEMI_MINOR_AXIS*Math.sin(par1);

        double x2 = SEMI_MAJOR_AXIS*Math.cos(par2)*Math.cos(lon2r);
        double y2 = SEMI_MAJOR_AXIS*Math.cos(par2)*Math.sin(lon2r);
        double z2 = SEMI_MINOR_AXIS*Math.sin(par2);

        // Compute the chord length [nm]
        double k = Math.sqrt(Math.pow(x1-x2,2)+Math.pow(y1-y2,2)+Math.p...
...ow(z1-z2,2));

        // Compute a correction factor
        double latmid = lat1r + (lat2r-lat1r)*0.5;
        double az = Math.atan2(Math.cos(lat2r)*Math.sin(lon2r-lon1r),
                Math.cos(lat1r)*Math.sin(lat2r)-Math.sin(lat1r)*Math.co...
...s(lat2r)
                *Math.cos(lon2r-lon1r));

        double rho = (SEMI_MAJOR_AXIS*(1-ECCENTRICITY*ECCENTRICITY))/
                Math.sqrt(Math.pow(1-Math.pow(ECCENTRICITY*Math.sin(lat...
...mid),2), 3));

        double nu = SEMI_MAJOR_AXIS /
                Math.sqrt(1-Math.pow(ECCENTRICITY*Math.sin(latmid),2));...
...
        // Apply correction factor
        double r = (rho*nu)/(rho*Math.pow(Math.sin(az),2)+nu*Math.pow(M...
...ath.cos(az),2));
        double delta = Math.pow(k,3)/(24*r*r) + 3*Math.pow(k,5)/(640*Ma...
...th.pow(r,4));
        distance_ = k + delta;

        azimuth_ = Math.toDegrees(az);
        if(azimuth_ < 0) azimuth_ += 360.0D;
        return this;
    }
```

```
    public double getDistance(){
        this.calculateDistance();
        return distance_;
    }

    public double getAzimuth(){
        return azimuth_;
    }

//    function rng = shortgeodesicdist(lat1, lon1, lat2, lon2, ellipsoi...
...d)
//
//% Calculate an approximate geodesic distance between nearby points on...
... an
//% ellipsoid.  LAT1, LON1, LAT2, and LON2 are in radians.  RNG is a le...
...ngth
//% and has the same units as the semi-major axis of the ellipsoid.
//
//a = ellipsoid(1);              %  Semimajor axis
//b = minaxis(ellipsoid);        %  Semiminor axis
//
//% Compute the Cartesian coordinates of the points on the ellipsoid
//% using their parametric latitudes.
//par1 = geod2par(lat1,ellipsoid,'radians');
//par2 = geod2par(lat2,ellipsoid,'radians');
//
//[x1,y1,z1] = sph2cart(lon1,par1,a);
//z1 = (b/a) * z1;
//
//[x2,y2,z2] = sph2cart(lon2,par2,a);
//z2 = (b/a) * z2;
//
//% Compute the chord length.  Can't use norm function because
//% x1, x2, etc may already be vectors or matrices.
//
//k = sqrt( (x1-x2).^2 +  (y1-y2).^2 +  (z1-z2).^2 );
//
//% Compute a correction factor, and then the range. The correction fac...
...tor
//% breaks down as the distance and/or the eccentricity increase.
//
//r = rsphere('euler',lat1,lon1,lat2,lon2,ellipsoid,'radians');
//delta = k.^3 ./ (24*r.^2) + 3*k.^5 ./ (640*r.^4);
//rng = k + delta;

}
```

MatlabDriver.java

```
/*
 * Java class to the MatrixDriver example from Java.
 * This class assumes a shared lib called
 * (lib)jnimatrixdriver.(so,dll) that contians the
 * necessary gateway functions for initialization,
 * termination, and execution of the compiled m-functions.
 * This shared library in turn links against a compiler
 * generated shared lib (libmatrix) that contians the actual
 * m-functions.
 */
public class MatlabDriver
{
    /*
     * Native methods corresponding to the functions
     * that we need to call in the compiler-generated
     * shared library.
     */
    public static native void applicationInitialize();
    public static native void libInitialize();
    public static native double[][] polyboolIntersection(double[][] a, ...
...double[][] b);
    public static native double[][] polyboolUnion(double[][] coord1, do...
...uble[][] coord2);
    public static native double[][] buffermOut(double[][] coord, double...
...[][] dist);
    public static native double[][] buffermIn(double[][] coord, double[...
...][] dist);
    public static native double[][] reckon2(double[][] coord, double[][...
...] rngAz);
    public static native double[][] meanm2(double[][] coord);
    public static native double[][] areaintNm(double[][] coord);
    public static native double[][] multiBufferm(double[][] coordIn1,
        double[][] coordIn2, double[][] distIn, double[][] isOut);
    public static native double[][] completeCluster(double[][] flCoordC...
...luster);
    public static native void libTerminate();
    public static native void applicationTerminate();
    public MatlabDriver()
    {
    }
    /* Helper function to print out a double matrix */
    public static void display(double[][] a)
    {
        for (int i=0; i<a.length; i++)
        {
            double[] row = a[i];
```

```
            for (int j=0; j<row.length; j++)
            {
                System.out.print(row[j] + " ");
            }
            System.out.print("\n");
        }
        System.out.print("\n");
    }
}
```

Network.java

```
/*
 * Network.java
 *
 * Created on June 6, 2007, 2:44 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author jeff
 */
import java.util.*;
import java.util.concurrent.*;
import java.io.*;
public class Network {
    /** Network representation based on flight plan. Link characteristi...
...cs vary
     *  by time period but this data is included in the scenario.
     *  Node characteristics are static across all flight plan time per...
...iods.
     *
     *  This data will include both active flights (TZ + Flight Plan) a...
...nd
     *  flights planning to depart during the analysis time period.
     */
    private ConcurrentHashMap<String,Node3D<Double>> nodes_;
    private ConcurrentHashMap<String,ArrayList<String>> nodesByIdentifi...
...er_;
    private ConcurrentHashMap<String,Link3D> links_;

    /* Maintains a list of links that are added (existing) to the netwo...
...rk. Key
     * is the from node id and the value is an arraylist of to node ids...
....
     */
    private ConcurrentHashMap<String, ArrayList<String>> existingNetwor...
...kLinks_;

    /* List of flight plan, weather, and demand scenarios.
     */
    private static final String flightPlanScenarioName = "Flight Plan";...
...    private ConcurrentHashMap<String, Scenario> scenarios_;
    private ArrayList<Scenario> sortedScenarios_;
```

```
    private ArrayList<ArrayList<Scenario>> groupedScenarios_;

    /* Constants for network
     */
    private final int NUM_SCENARIOS = 22;
    private final int NUM_WEATHER_SCENARIOS = 20;
    private final int NUM_NODES = 250000;
    private final int NUM_LINKS = 400000;
    private final int NUM_NODES_BY_IDENT = 100000;
    private final int NUM_LINKS_BY_FROMNODE = 225000;
    private final int NUMSCENARIOSPERGROUP = 5;


    /* Sectors that are capacity constrained
     */
    private ArrayList<ASector> constrainedSectors_;

    /* Flights passing through the flow constrained area
     */
    private ArrayList<Flight> constrainedFlightsLessThan18kft_;
    private ArrayList<Flight> constrainedFlightsGreater18kft_;


    /** Creates a new instance of Network */
    public Network() {
        nodes_ = new ConcurrentHashMap<String,Node3D<Double>>(
                NUM_NODES,
                Settings.hashMapLoadFactor, 4);
        nodesByIdentifier_ = new ConcurrentHashMap<String,ArrayList<Str...
...ing>>(
                NUM_NODES_BY_IDENT,
                Settings.hashMapLoadFactor, 4);
        links_ = new ConcurrentHashMap<String,Link3D>(
                NUM_LINKS,
                Settings.hashMapLoadFactor, 4);
        existingNetworkLinks_ = new ConcurrentHashMap<String, ArrayList...
...<String>>(
                NUM_LINKS_BY_FROMNODE,
                Settings.hashMapLoadFactor, 4);
        scenarios_ = new ConcurrentHashMap<String, Scenario>(
                NUM_SCENARIOS,
                Settings.hashMapLoadFactor, 4);

        // Initialize the flight plan scenario
        Scenario flightPlanScenario = new Scenario(flightPlanScenarioNa...
...me,
                this, APData.airspace);
```

```
        scenarios_.put(flightPlanScenarioName, flightPlanScenario);

        // Initialize weather scenarios
        for(int i=0; i < 20; i++){
            String weatherScenarioName = "Weather " + Integer.toString(...
...i+1);
            Scenario weatherScenario = new Scenario(weatherScenarioName...
..., this,
                    APData.airspace);
            scenarios_.put(weatherScenarioName, weatherScenario);
        }

        constrainedSectors_ = new ArrayList<ASector>();

        constrainedFlightsLessThan18kft_ = new ArrayList<Flight>();
        constrainedFlightsGreater18kft_ = new ArrayList<Flight>();
    }


    public double[] getConstrainedSectorsLat(){
        int totalNumberNodes = 0;
        int totalSectorModules = 0;

        for(ASector sector:constrainedSectors_){
            for(int i=0; i<sector.noModules(); i++){
                totalSectorModules++;
                ASectorModule module = sector.getModule(i);
                totalNumberNodes += module.noNodes();
            }
        }

        double[] lats = new double[totalNumberNodes+totalSectorModules]...
...;
        int k = 0;
        for(ASector sector:constrainedSectors_){
            for(int i=0; i<sector.noModules(); i++){
                ASectorModule module = sector.getModule(i);
                for(int j=0; j<module.noNodes(); j++){
                    GPoint2D<Double> pt = module.getNode(j);
                    lats[k] = pt.getLatitude();
                    k++;
                }
                lats[k] = Double.NaN;
                k++;
            }
        }
```

```
        return lats;
    }


    public double[] getConstrainedSectorsLon(){
        int totalNumberNodes = 0;
        int totalSectorModules = 0;

        for(ASector sector:constrainedSectors_){
            for(int i=0; i<sector.noModules(); i++){
                totalSectorModules++;
                ASectorModule module = sector.getModule(i);
                totalNumberNodes += module.noNodes();
            }
        }

        double[] lons = new double[totalNumberNodes+totalSectorModules]...
...;
        int k = 0;
        for(ASector sector:constrainedSectors_){
            for(int i=0; i<sector.noModules(); i++){
                ASectorModule module = sector.getModule(i);
                for(int j=0; j<module.noNodes(); j++){
                    GPoint2D<Double> pt = module.getNode(j);
                    lons[k] = pt.getLongitude();
                    k++;
                }
                lons[k] = Double.NaN;
                k++;
            }
        }

        return lons;
    }

    public Network calculateFlightsInFlowConstrainedArea(){
        Airspace a = APData.airspace;

        constrainedFlightsLessThan18kft_ = new ArrayList<Flight>();
        constrainedFlightsGreater18kft_ = new ArrayList<Flight>();

        Scenario flightPlanScenario = getFlightPlanScenario();

        ArrayList<String> constrainedFlights =
                flightPlanScenario.getFlightsThroughFlowConstrainedArea...
...(constrainedSectors_);
```

```
        for(String flightId:constrainedFlights){
            Flight flight = a.getFlight(flightId);

            int flightLevel = flight.getCruisingFlightLevel();

            if(flightLevel < 180){
                constrainedFlightsLessThan18kft_.add(flight);
            }else{
                constrainedFlightsGreater18kft_.add(flight);
            }
        }

        // Remove demand from flow constrained flights for weather scen...
...arios
        ConcurrentHashMap<String, HashMap<Integer, Double>> flightDeman...
...dTimePeriod =
                flightPlanScenario.getFlightDemandByTimePeriod();
        for(int i=0; i<NUM_WEATHER_SCENARIOS; i++){
            Scenario wxScenario = getWeatherScenario(i);
            wxScenario.removeDemandFromConstrainedFlights(constrainedFl...
...ights,
                    flightDemandTimePeriod);
        }

        return this;
    }

    public ArrayList<Flight> getConstrainedFlightsLessThan18kft(){
        return constrainedFlightsLessThan18kft_;
    }

    public ArrayList<Flight> getConstrainedFlightsGreater18kft(){
        return constrainedFlightsGreater18kft_;
    }

    public Network copyFlightPlanDemandToWeatherScenarios(){
        Scenario flightPlanScenario = getFlightPlanScenario();

        double[][] sectorDemand = flightPlanScenario.getSectorDemandArr...
...ay();

        for(int i=0; i<NUM_WEATHER_SCENARIOS; i++){
            Scenario wxScenario = getWeatherScenario(i);

            wxScenario.initializeSectorCapacityAndFlow();
            wxScenario.setSectorDemandArray(sectorDemand);
        }
```

```
        return this;
    }

    public ArrayList<ASector> flowConstrainedSectors(){
        return constrainedSectors_;
    }

    public ArrayList<ASector> calculateFlowConstrainedSectors(){
        /** Statically typed sectors to exclude */
        String[] excludeS = Settings.sectorsToExclude.split(",");
        ArrayList<String> sectorsToExclude = new ArrayList<String>(
                Arrays.asList(excludeS));

        int startTimeMin = (Settings.analysisTimeFilterStart + 1)*60;
        int endTimeMin   = (Settings.analysisTimeFilterEnd   + 2)*60;

        constrainedSectors_ = new ArrayList<ASector>();

        for(int i=0; i<NUM_WEATHER_SCENARIOS; i++){
            Scenario wxScenario = getWeatherScenario(i);

            ArrayList<ASector> wxConstrainedSectors =
                    wxScenario.calculateFlowConstrainedSectors(startTim...
...eMin, endTimeMin);

            for(ASector sector:wxConstrainedSectors){
                // Exclude sectors based on sector and center identifie...
...rs in
                // sectorsToExclude.
                String sectorS = sector.getName();
                if(sectorsToExclude.contains(sectorS) || sectorS.length...
...() < 3) continue;
                String centerS = sectorS.substring(0,3);
                if(sectorsToExclude.contains(centerS)) continue;

                if(!constrainedSectors_.contains(sector)) constrainedSe...
...ctors_.add(sector);
            }
        }


        return constrainedSectors_;
    }

    public Network printHashMapSizes(){
        System.out.println("nodes_ : " + Integer.toString(nodes_.size()...
```

```
...));
        System.out.println("nodesByIdentifier_ : " + Integer.toString(n...
...odesByIdentifier_.size()));
        System.out.println("links_ : " + Integer.toString(links_.size()...
...));
        System.out.println("existingNetworkLinks_ : " + Integer.toStrin...
...g(existingNetworkLinks_.size()));

        return this;
    }

    public Scenario getFlightPlanScenario(){
        return scenarios_.get(flightPlanScenarioName);
    }

    public int getNumberWeatherScenarios(){
        return NUM_WEATHER_SCENARIOS;
    }

    public Scenario getWeatherScenario(int i){
        String weatherScenarioName = "Weather " + Integer.toString(i+1)...
...;
        return scenarios_.get(weatherScenarioName);
    }

    // Add flight plan links to the first time period for now
    public synchronized Network addLink(String fromId, String toId,
            double floorAltitude, double ceilingAltitude, Link3D.LinkTy...
...pes linkType){
        // Create identifier using the nodes
        StringBuilder s = new StringBuilder(fromId);
        s.append(APData.BAR);
        s.append(toId);
        String linkIdentifier = s.toString();
        s = null;

        // Do not add duplicate network links
        if(this.hasLink(linkIdentifier)) return this;

        // Get node objects
        Node3D<Double> fromNode = nodes_.get(fromId);
        Node3D<Double> toNode = nodes_.get(toId);

        // Create link
        Link3D link = new Link3D(linkIdentifier, fromNode, fromId, toNo...
...de, toId, linkType);
```

```
        if(link.isAirportConnector() && toNode.getParentSector().equals...
...(ASector.NullSector)){
            // Do not exclude airport links to external nodes
        }else if(linkIdentifier.contains(APData.FORWARDSLASH)){
            // Do not exclude links with 3200N/06000W
        }else if(fromNode.getParentSector().equals(ASector.NullSector) ...
...&&
                toNode.getParentSector().equals(ASector.NullSector)){
            // Do not check distance for external links
        }else{

            // Check for breaks caused by domestic and international ai...
...rways sharing
            // the same identifier.
            boolean sameSigns = sameSign(fromNode.getLongitude(), toNod...
...e.getLongitude()) &&
                    sameSign(fromNode.getLatitude(), toNode.getLatitude...
...());
            double absLongDiff = Math.abs(fromNode.getLongitude() - toN...
...ode.getLongitude());

            // Case where -180d = 180d
            if(Math.abs(fromNode.getLongitude()) > 175.0D &&
                    Math.abs(toNode.getLongitude()) > 175.0D) absLongDi...
...ff = 0.0D;

            double absLatDiff = Math.abs(fromNode.getLatitude() - toNod...
...e.getLatitude());
            if(!sameSigns && (absLongDiff > 15.0D || absLatDiff > 10.0D...
...)){
                return this;
            }else if(absLongDiff > 15.0D || absLatDiff > 10.0D){
                return this;
            }
        }


        // Store the link
        links_.put(linkIdentifier, link);

        // Set the parent sector for the link
        link.setParentSector(fromNode.getParentSector());

        // Set altitude limits for the link
        link.setAltitudeLimits(floorAltitude, ceilingAltitude);

        // Add the link to the from node out link list
```

```
        fromNode.addLinkToOutLinkList(link, linkIdentifier);

        // Add to list of existing network links using original identif...
...iers
        String fromId0 = fromNode.getAirspaceIdentifier();
        String toId0 = toNode.getAirspaceIdentifier();
        ArrayList<String> existNetLinks = existingNetworkLinks_.contain...
...sKey(fromId0) ?
            existingNetworkLinks_.get(fromId0) : new ArrayList<String>(...
...5);
        existNetLinks.add(toId0);
        existingNetworkLinks_.put(fromId0, existNetLinks);

        // Add to list of existing network links using node identifiers...
...        existNetLinks = existingNetworkLinks_.containsKey(fromId) ?
            existingNetworkLinks_.get(fromId) : new ArrayList<String>(5...
...);
        existNetLinks.add(toId);
        existingNetworkLinks_.put(fromId, existNetLinks);

        linkIdentifier = null;

        return this;
    }

// Check to see if a potential link exists (i.e. flight plans may have
// additional links. This code works for both the original identifiers ...
...and
// the modified node identifiers.
    public boolean hasLink(String fromId, String toId){
        if(existingNetworkLinks_.containsKey(fromId)){
            ArrayList<String> existNetLinks = existingNetworkLinks_.get...
...(fromId);
            if(existNetLinks.contains(toId)){
                return true;
            }
        }

        // No matching link is found
        return false;
    }

// Check to see if link is in the network
    public boolean hasLink(String linkId){
        return links_.containsKey(linkId);
    }
```

```
    public boolean sameSign(double val1, double val2){
        if(val1 <= 0 && val2 <= 0){
            return true;
        }else if(val1 > 0 && val2 > 0){
            return true;
        }else{
            return false;
        }

    }

    public Link3D getLink(String linkId){
        return links_.get(linkId);
    }

    public Iterator<String> getLinkIterator(){
        return links_.keySet().iterator();
    }

    public Iterator<Link3D> getLinkValueIterator(){
        return links_.values().iterator();
    }

    public int noLinks(){
        return links_.size();
    }

//
    public Network addNode(String identifier, Node3D<Double> location){...
...
        // Add notes
        if(!nodes_.containsKey(identifier)){
            nodes_.put(identifier, location);
        }

        // Add to list of nodes by airspace identifier
        String airspaceIdentifier = location.getAirspaceIdentifier();

        /** Adds a reference to a node e.g. "AAA_FLOOR_180" using the
         *  identifier "AAA" as a key. */
        if(nodesByIdentifier_.containsKey(airspaceIdentifier)){
            ArrayList<String> fpNodes = nodesByIdentifier_.get(airspace...
...Identifier);
            if(!fpNodes.contains(identifier)) fpNodes.add(identifier);
        }else{
            ArrayList<String> fpNodes = new ArrayList<String>(5);
            fpNodes.add(identifier);
```

```
            nodesByIdentifier_.put(airspaceIdentifier, fpNodes);
        }

        return this;
    }

// Remove node
    public Network removeNode(String identifier){
        if(nodes_.containsKey(identifier)){
            Node3D<Double> location = nodes_.get(identifier);
            nodes_.remove(identifier);

            String airspaceIdentifier = location.getAirspaceIdentifier(...
...);
            if(nodesByIdentifier_.containsKey(airspaceIdentifier)){
                ArrayList<String> fpNodes = nodesByIdentifier_.get(airs...
...paceIdentifier);
                fpNodes.remove(identifier);
            }
        }
        return this;
    }


    /** Iterator to identifiers used to build flight plan nodes. */
    public Iterator<String> getIdentifierIterator(){
        return nodesByIdentifier_.keySet().iterator();
    }

    /** List of flight plan nodes using the specified identifier. */
    public ArrayList<String> getNodesUsingIdentifier(String identifier)...
...{
        return nodesByIdentifier_.get(identifier);
    }

    /** Checks if identifier is currently used by one or more nodes */
    public boolean hasNode(String identifier){
        return nodesByIdentifier_.containsKey(identifier);
    }

    /** Iterator to flight plan node identifiers. */
    public Iterator<String> getNodeIterator(){
        return nodes_.keySet().iterator();
    }

    /** Get list of all nodes and airports */
    public ArrayList<Node3D<Double>> getNodeAirportSet(){
```

```
        ArrayList<Node3D<Double>> nodesToReturn = null;
        int noNodes = nodes_.size() + APData.airspace.noAirports();
        nodesToReturn = new ArrayList<Node3D<Double>>(noNodes);

        for(Node3D<Double> node:nodes_.values()){
            nodesToReturn.add(node);
        }

        Iterator<Airport> e = APData.airspace.getAirportValueIterator()...
...;
        while(e.hasNext()){
            nodesToReturn.add(e.next());
        }

        return nodesToReturn;
    }

    public ArrayList<Node3D<Double>> getNodeCollectionByNanIndex(int in...
...dex){
        ArrayList<Node3D<Double>> nodesToReturn = null;

        // Count the number of nodes that are not NaN at the index spec...
...ified
        int noNodes = 0;
        for(Node3D<Double> node:nodes_.values()){
            if(!Double.isNaN(node.getDoubleValue(index))){
                noNodes++;
            }
        }

        // Get non-NaN airports
        Airspace a = APData.airspace;
        Iterator<Airport> e = a.getAirportValueIterator();
        while(e.hasNext()){
            Airport apt = e.next();
            if(!Double.isNaN(apt.getDoubleValue(index))){
                noNodes++;
            }
        }

        nodesToReturn = new ArrayList<Node3D<Double>>(noNodes);

        // Add the nodes that are not NaN at the index specified
        for(Node3D<Double> node:nodes_.values()){
            if(!Double.isNaN(node.getDoubleValue(index))){
                nodesToReturn.add(node);
            }
```

```
        }

        // Add non-NaN airports
        e = a.getAirportValueIterator();
        while(e.hasNext()){
            Airport apt = e.next();
            if(!Double.isNaN(apt.getDoubleValue(index))){
                nodesToReturn.add(apt);
            }
        }

        return nodesToReturn;
    }

    public Iterator<Node3D<Double>> getNodeValueIterator(){
        return nodes_.values().iterator();
    }

    /** Get flight plan node based on identifier */
    public Node3D<Double> getNode(String identifier){
        return nodes_.get(identifier);
    }

    public int noNodes(){
        return nodes_.size();
    }

// Trim ArrayLists after creation to reduce storage space
    public Network trimArrayLists(){
        Iterator<Node3D<Double>> e = nodes_.values().iterator();
        while(e.hasNext()){
            Node3D<Double> node = e.next();
            node.trimArrayLists();
        }

        Iterator<ArrayList<String>> e2 = this.nodesByIdentifier_.values...
...().iterator();
        while(e2.hasNext()){
            ArrayList<String> al = e2.next();
            al.trimToSize();
        }

        Iterator<ArrayList<String>> e3 = this.existingNetworkLinks_.val...
...ues().iterator();
        while(e3.hasNext()){
            ArrayList<String> al = e3.next();
            al.trimToSize();
```

```
        }

        return this;
    }

    /** Set the floor of the lowest links to zero so that flights do no...
...t attempt
     * to descend below the lowest link. */
    public Network setFloorLowestLinksToGround(){
        for(Node3D<Double> node:nodes_.values()){

            // Not needed for external sectors
            if(node.getParentSector().equals(ASector.NullSector)){
                continue;
            }

            double minCeilingNodeAltitude = 999.0D;
            double minCentroidNodeAltitude = 999.0D;
            double minFloorNodeAltitude = 999.0D;

            Node3D<Double> minCeilingNode = null;
            Node3D<Double> minCentroidNode = null;
            Node3D<Double> minFloorNode = null;

            String minCentroidNodeId = null;

            // Get the lowest node
            ArrayList<String> otherIds = this.getNodesUsingIdentifier(n...
...ode.getAirspaceIdentifier());
            for(String nodeId2:otherIds){
                Node3D<Double> node2 = nodes_.get(nodeId2);

                if(node2.isAtCeiling() && node2.getElevation() < minCei...
...lingNodeAltitude){
                    minCeilingNodeAltitude = node2.getElevation();
                    minCeilingNode = node2;
                }else if(node2.isAtCentroid() && node2.getElevation() <...
... minCentroidNodeAltitude){
                    minCentroidNodeAltitude = node2.getElevation();
                    minCentroidNode = node2;
                    minCentroidNodeId = nodeId2;
                }else if(node2.isAtFloor()){
                    minFloorNodeAltitude = node2.getElevation();
                    minFloorNode = node2;
                }
            }
```

```
            // Set the stored altitude to 0
            ASector minSector = minCentroidNode.getParentSector();
            String airspaceId = minCentroidNode.getAirspaceIdentifier()...
...;
            if(minSector.hasFix(airspaceId)){
                double ceilingAltitude = minSector.getFixModuleCeiling(...
...airspaceId);
                minSector.addFixToSector(airspaceId, 0.0D, ceilingAltit...
...ude);
            }
            if(minSector.hasNavaid(airspaceId)){
                double ceilingAltitude = minSector.getNavaidModuleCeili...
...ng(airspaceId);
                minSector.addNavaidToSector(airspaceId, 0.0D, ceilingAl...
...titude);

            }

            // Only set elevation for links from ceiling to centroid
            if(minCeilingNode != null){
                for(Link3D outLink:minCeilingNode.getOutLinkList()){
                    if(outLink.getToNode().isAtCentroid()){
                        outLink.setAltitudeLimits(0.0D, outLink.getCeil...
...ingAltitude());
                    }
                }
            }

            // Set the elevation for all outlinks from centroid
            if(minCentroidNode != null){
                for(Link3D outLink:minCentroidNode.getOutLinkList()){
                    outLink.setAltitudeLimits(0.0D, outLink.getCeilingA...
...ltitude());
                }

                // Search for links from a boundary node to the lowest ...
...centroid
                for(String nodeId2:otherIds){
                    Node3D<Double> node2 = nodes_.get(nodeId2);

                    if(!node2.isAtBoundary()) continue;

                    ArrayList<Link3D> outLinks = node2.getOutLinkList()...
...;

                    for(Link3D link:outLinks){
                        if(link.getToNodeId().equals(minCentroidNodeId)...
```

```
...){
                            link.setAltitudeLimits(0.0D, link.getCeilin...
...gAltitude());
                        }
                    }
                }
            }

            // Only set elevation for links from floor to centroid
            if(minFloorNode != null){
                for(Link3D outLink:minFloorNode.getOutLinkList()){
                    if(outLink.getToNode().isAtCentroid()){
                        outLink.setAltitudeLimits(0.0D, outLink.getCeil...
...ingAltitude());
                    }
                }
            }
        }

        return this;
    }

    // Thread class to assign flights to network
    private class assignPlannedFlightsNetwork implements Runnable{
        private Scenario s_;
        private Flight f_;

        public assignPlannedFlightsNetwork(Scenario s, Flight f){
            s_ = s;
            f_ = f;
        }

        public void run(){
            boolean assignmentOk = s_.assignFlightDemand(f_);
            if(!assignmentOk) errors++;
        }
    }

    // Keep track of errors
    private int errors;

    public Network assignPlannedFlights(){
        Airspace a = APData.airspace;

        Scenario flightPlanScenario = getFlightPlanScenario();

        // Initialize sector capacity and flow to 0
```

```
        flightPlanScenario.initializeSectorCapacityAndFlow();

        // Keep track of errors in assignment
        int totalFlights = 0;
        errors = 0;
        //boolean assignmentOk = true;

        // ArrayList to keep track of thread
        ArrayList<Thread> threads = new ArrayList<Thread>(a.noFlights()...
...);

        // Iterate through each flight
        Iterator<String> e = a.getFlightIterator();
        //int maxIteration = 0;
        while(e.hasNext()){
            String flightId = e.next();
            Flight f = a.getFlight(flightId);

            // Create the thread
            Thread tAssign = new Thread(new assignPlannedFlightsNetwork...
...(flightPlanScenario, f));
            tAssign.start();
            threads.add(tAssign);

            totalFlights++;

            //if(maxIteration++ > 1000) break;

            if(totalFlights % 100 == 0){
                try{
                    tAssign.join();
                }catch(InterruptedException e3){
                    System.out.println("Error assigning planned flights...
...!");
                }

                if(totalFlights % 1000 == 0){
                    System.out.println("     " + Integer.toString(total...
...Flights) + " flights assigned");
                }
            }
        }

        for(Thread tAssign:threads){
            try{
                tAssign.join();
            }catch(InterruptedException e2){
```

```
                System.out.println("Assignment of planned flights inter...
...rupted!");
            }
        }

        double totalDemand = flightPlanScenario.totalSectorDemand();

        System.out.println("Flights: " + Integer.toString(totalFlights)...
...);
        System.out.println("Errors : " + Integer.toString(errors));

        System.out.println("Total demand (aircraft-minutes): " + Double...
....toString(totalDemand));

        return this;
    }

    // Rank scenarios by capacity loss - then group by impact
    public Network rankScenariosByAirspaceImpact(){
        // Sort the scenarios
        ArrayList<Scenario> sortedWxScenarios =
                new ArrayList<Scenario>(getNumberWeatherScenarios());

        int i;
        for(i=0; i<getNumberWeatherScenarios(); i++){
            sortedWxScenarios.add(getWeatherScenario(i));
        }

        // Scenarios with the lowest index are highest priority
        // i.e. have the largest
        Collections.sort(sortedWxScenarios);

        sortedScenarios_ = sortedWxScenarios;

        // Create a list of scenarios in each group
        int groupCount = 0;
        ArrayList<Scenario> scenariosInGroup = new ArrayList<Scenario>(...
...NUMSCENARIOSPERGROUP);
        groupedScenarios_ = new ArrayList<ArrayList<Scenario>>();
        for(i=0; i<sortedScenarios_.size(); i++){
            groupCount++;
            scenariosInGroup.add(sortedScenarios_.get(i));
            if(groupCount >= 5){
                groupCount = 0;
                groupedScenarios_.add(scenariosInGroup);
                scenariosInGroup = new ArrayList<Scenario>(NUMSCENARIOS...
...PERGROUP);
```

```
            }
        }

        return this;
    }

    // Group index is zero-based
    public ArrayList<Scenario> getScenarioGroup(int groupIndex){

        ArrayList<Scenario> scenariosToReturn = new ArrayList<Scenario>...
...(
                NUMSCENARIOSPERGROUP);

        int lowerIdx = groupIndex * NUMSCENARIOSPERGROUP;
        int upperIdx = (groupIndex + 1) * NUMSCENARIOSPERGROUP - 1;

        int i;
        for(i = lowerIdx; i <= upperIdx; i++){
            scenariosToReturn.add(sortedScenarios_.get(i));
        }

        return scenariosToReturn;
    }

    public Collection<Node3D<Double>> getNodeSet(){
        return nodes_.values();
    }

    public Collection<Link3D> getLinkSet(){
        return links_.values();
    }

    /** Calculate the length of each link that is not
        already set. Links that are not set are
     *  initialized to an invalid length (e.g. -1).
     */
    public Network calculateLinkLengths(){
        for(Link3D link:links_.values()){
            if(link.getLength() < 0){
                Node3D<Double> fromNode = link.getFromNode();
                Node3D<Double> toNode = link.getToNode();

                double lat1 = fromNode.getLatitude();
                double lon1 = fromNode.getLongitude();

                double lat2 = toNode.getLatitude();
                double lon2 = toNode.getLongitude();
```

```
            Mapping m = new Mapping(lat1, lon1, lat2, lon2);
            //m.calculateDistance();

            link.setLength(m.getDistance());
        }
    }

    return this;
}

public Network resetNodeLinkDoubleValues(int index){
    // Reset the links
    for(Link3D link:this.getLinkSet()){
        link.setDoubleValue(index, 0.0D);
    }

    // Reset the nodes
    for(Node3D<Double> node:this.getNodeAirportSet()){
        node.setDoubleValue(index, 0.0D);
        node.setIntegerValue(index, 0);
    }

    return this;
}

public Network initializeNodeLinkDoubleValues(int numValues){
    // Initialize the links
    int i;
    for(Link3D link:this.getLinkSet()){
        link.initializeDoubleValues(numValues);

        for(i=0; i<numValues; i++){
            link.setDoubleValue(i, 0.0D);
        }
    }

    // Initialize the nodes
    for(Node3D<Double> node:this.getNodeAirportSet()){
        node.initializeDoubleValues(numValues);
        node.initializeIntegerValues(numValues);

        for(i=0; i<numValues; i++){
            node.setDoubleValue(i, 0.0D);
            node.setIntegerValue(i, 0);
        }
    }
```

```
        return this;
    }

    public Network copyNodeLinkDoubleValues(int fromLabelIndex, int toL...
...abelIndex){

        for(Link3D link:this.getLinkSet()){
            link.setDoubleValue(toLabelIndex, link.getDoubleValue(fromL...
...abelIndex));
        }

        for(Node3D<Double> node:this.getNodeAirportSet()){
            node.setDoubleValue(toLabelIndex, node.getDoubleValue(fromL...
...abelIndex));
        }

        return this;
    }

    // This is part of the pre-processing for the APCDM k-shortest path...
...s. If the
    // sector has zero capacity then remove all of the links in the sec...
...tor from
    // consideration.
    public Network removeLinksInSectorsWithZeroCapacity(int index){
        // Get the weather forecast scenario
        Scenario wxForecastScenario = this.getWeatherScenario(0);

        // Get a list of sectors with zero capacity
        int analysisTimePeriod = (Settings.analysisTimeFilterStart + 1)...
...*4;
        ArrayList<ASector> sectorsWithZeroCapacity = new ArrayList<ASec...
...tor>();

        for(ASector sector:this.constrainedSectors_){
            double secCapacity = wxForecastScenario.getSectorCapacity15...
...Min(
                    sector.getArrayIndex(), analysisTimePeriod);

            if(secCapacity < 1.0D) sectorsWithZeroCapacity.add(sector);...
...        }

        // Find links that are in the sectors with zero capacity
        for(Link3D link:this.getLinkSet()){
            ASector sector = link.getParentSector();
```

```
            if(sectorsWithZeroCapacity.contains(sector)){
                link.setDoubleValue(index, Double.MAX_VALUE);
            }
        }

        return this;
    }

    // Thread class for incremental assignment of flights to the networ...
...k
    private class assignmentIncremental implements Runnable {
        public int j;
        private int i2;
        public ArrayList<Link3D> spaPath;
        private Network network_;
        private final int startTimeMin = (Settings.analysisTimeFilterSt...
...art + 1) * 60;
        private final int endTimeMin = (Settings.analysisTimeFilterEnd ...
...+ 2) * 60;
        private ArrayList<String> sectorsToExclude;
        private int labelIndexLt18k;
        private int labelIndexGt18k;
        private double penaltyAltitudeOutOfRange;
        private ArrayList<Scenario> weatherScenarios;
        public assignmentIncremental(int j_, int i2_, Network network,
                ArrayList<String> sectorsToExclude_,
                int labelIndexLt18k_, int labelIndexGt18k_,
                double penaltyAltitudeOutOfRange_,
                ArrayList<Scenario> weatherScenarios_) {
            j = j_;
            i2 = i2_;
            network_ = network;
            sectorsToExclude = sectorsToExclude_;
            labelIndexLt18k = labelIndexLt18k_;
            labelIndexGt18k = labelIndexGt18k_;
            penaltyAltitudeOutOfRange = penaltyAltitudeOutOfRange_;
            weatherScenarios = weatherScenarios_;
        }
        public void run() {
            // Use the weather forecast scenario for initial calculatio...
...ns
            Scenario flightPlanScenario = network_.getFlightPlanScenari...
...o();
            // Reset the link travel time
            resetNodeLinkDoubleValues(j);
            if (i2 > Settings.numberFlightClustersLT18kft) {
                copyNodeLinkDoubleValues(labelIndexGt18k, j);
```

```
            } else {
                copyNodeLinkDoubleValues(labelIndexLt18k, j);
            }

            // Iterate through each of the weather scenarios in the wea...
...ther scenario group
            ArrayList<ASector> wxGrpConstrainedSectors = new ArrayList<...
...ASector>();
            for (Scenario wxScenario : weatherScenarios) {
                ArrayList<ASector> wxConstrainedSectors =
                        wxScenario.calculateFlowConstrainedSectors(star...
...tTimeMin, endTimeMin);
                for (ASector sector : wxConstrainedSectors) {
                    // Exclude sectors based on sector and center ident...
...ifiers in
                    // sectorsToExclude.
                    String sectorS = sector.getName();
                    if (sectorsToExclude.contains(sectorS) || sectorS.l...
...ength() < 3) {
                        continue;
                    }
                    String centerS = sectorS.substring(0, 3);
                    if (sectorsToExclude.contains(centerS)) {
                        continue;
                    }
                    if (!wxGrpConstrainedSectors.contains(sector)) {
                        wxGrpConstrainedSectors.add(sector);
                    }
                }
            }
            // Do not consider links that are in the flow constrained s...
...ectors
            for (Link3D link : network_.getLinkSet()) {
                ASector sector = link.getParentSector();
                if (wxGrpConstrainedSectors.contains(sector)) {
                    link.setDoubleValue(j, Double.MAX_VALUE);
                }
            }
            // Find the shortest path for this cluster and weather grou...
...p combination
            ArrayList<ArrayList<Link3D>> spaResult =
                    flightPlanScenario.calculateKShortestPathsForCluste...
...r(
                    i2, 1, penaltyAltitudeOutOfRange, j);
            // There is only one path in the result
            spaPath = spaResult.get(0);
        }
```

```
    }



    // Perform incremental assignment for each cluster and each weather...
... scenario
    public Network incrementalAssignment(double penaltyAltitudeOutOfRan...
...ge){

        // Reset sector demand for the weather scenarios to zero
        for(int i=0; i<NUM_WEATHER_SCENARIOS; i++){
            Scenario wxScenario = getWeatherScenario(i);
            wxScenario.initializeSectorCapacityAndFlow();
        }

        /** Statically typed sectors to exclude */
        String[] excludeS = Settings.sectorsToExclude.split(",");
        ArrayList<String> sectorsToExclude = new ArrayList<String>(
                Arrays.asList(excludeS));

        int startTimeMin = (Settings.analysisTimeFilterStart + 1)*60;
        int endTimeMin   = (Settings.analysisTimeFilterEnd   + 2)*60;


        // Sort scenarios by airspace impact
        this.rankScenariosByAirspaceImpact();

        // Use one label index for each of the scenario groups
        int numScenarioGroups = groupedScenarios_.size();

        int labelIndexLt18k = numScenarioGroups;
        int labelIndexGt18k = labelIndexLt18k + 1;

        // Initialize the label values for the nodes and links
        initializeNodeLinkDoubleValues(numScenarioGroups+2);

        // Use the weather forecast scenario for initial calculations
        Scenario flightPlanScenario = this.getFlightPlanScenario();

        // Initialize graph for flights operating below 18kft
        // Use the first < 18kft cluster
        flightPlanScenario.createDiGraphForCluster(1, penaltyAltitudeOu...
...tOfRange,
                labelIndexLt18k);

        // Initialize graph for flights operating above 18kft
        // Use the first > 18kft cluster
```

```
        flightPlanScenario.createDiGraphForCluster(Settings.numberFligh...
...tClustersLT18kft + 1,
                penaltyAltitudeOutOfRange, labelIndexGt18k);

        // Reset some counters
        Scenario.numNullPaths = 0;
        Scenario.numPathCalculations = 0;

        // Iterate through each of the clusters
        for(int i=0; i<flightPlanScenario.noClusters(); i++){

            // Cluster index is 1 based
            int i2 = i+1;

            if(i2 == 70){
                int i3 = i2 + 1;
            }

            // Get the number of flights in this cluster
            ArrayList<String> flightsInCluster = flightPlanScenario.get...
...FlightsInCluster(i2);
            int noFlightsInCluster = flightsInCluster.size();

            // Create list of threads
            ArrayList<assignmentIncremental> assignmentList =
                    new ArrayList<assignmentIncremental>(numScenarioGro...
...ups);
            ArrayList<Thread> threadList = new ArrayList<Thread>(numSce...
...narioGroups);

            // Create a thread for each of the weather scenario groups
            for(int j=0; j<numScenarioGroups; j++){
                // Get list of weather scenarios in the group
                ArrayList<Scenario> weatherScenarios = groupedScenarios...
..._.get(j);

                assignmentIncremental assignmentIncrementalVar = new as...
...signmentIncremental(
                        j, i2, this, sectorsToExclude, labelIndexLt18k,...
...                        labelIndexGt18k, penaltyAltitudeOutOfRange,
                        weatherScenarios);
                assignmentList.add(assignmentIncrementalVar);

                Thread t = new Thread(assignmentIncrementalVar);
                threadList.add(t);

                // Start the thread
```

```
                t.start();

            }

            // Store the results for each group
            for(int j=0; j<numScenarioGroups; j++){
                try{
                    Thread t = threadList.get(j);
                    t.join();
                }catch(InterruptedException e){
                    System.out.println("Could not finish incremental as...
...signment calculations!");
                }

                // There is only one path in the result
                ArrayList<Link3D> spaPath = assignmentList.get(j).spaPa...
...th;

                // Get list of weather scenarios in the group
                ArrayList<Scenario> weatherScenarios = groupedScenarios...
..._.get(j);

                // Assign flights and update demand
                flightPlanScenario.incrementDemandFromIncrementalAssign...
...ment(
                        i2, spaPath, weatherScenarios, j);
            }
            // Don't consider all of the clusters for now
            //if(i2 > 100) break;
            if(i2 % 10 == 0){
                System.out.print("Incremental Assignment - Cluster ");
                System.out.print(i2);
                System.out.print(" , Calcs = ");
                System.out.print(Scenario.numPathCalculations);
                System.out.print(" , Null = ");
                System.out.println(Scenario.numNullPaths);
            }
        }

        return this;
    }

}
```

NetworkBuilder.java

```
/*
 * NetworkBuilder.java
 *
 * Created on June 6, 2007, 3:25 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author jeff
 */
import java.util.*;
public class NetworkBuilder {
    // Do not consider sectors with a lower altitude less than 0 ft
    // Replaces original code of 18,000 ft
    public static final double MINSECTORFLOORALT = 0.0D;

    // Do not consider sectors with an upper altitude less than 0 ft
    // Replaces original code of 18,000 ft
    public static final double MINSECTORCEILALT = 0.0D;

    // Default elevation for Navaids/Fixes outside sector boundaries
    public static final double DEFAULTNODEELEVATION = 200.0D;
    public static final double DEFAULTFIRSTNODEELEV = 50.0D;

    /** Creates a new instance of NetworkBuilder */
    public NetworkBuilder() {
    }

    // Thread class to create airport nodes
    private static class addAirportsToNodes2 implements Runnable{
        public void run(){
            Airspace a = APData.airspace;
            NetworkBuilder.addAirportsToNodes(a.getFlightIterator());
        }

    }

    // Thread class to create Navaid and Fix nodes
    private static class addNavaidsAndFixesToNodes2 implements Runnable...
...{
        public void run(){
            NetworkBuilder.addNavaidsAndFixesToNodes(FAPio.writeCenters...
...);
```

```
        }
    }

    // Thread class to create nodes that represent the intersection of ...
...airways
    // and sector boundaries
    private static class addAirwaySectorIntersectionsToNodes2 implement...
...s Runnable{
        public void run(){
            NetworkBuilder.addAirwaySectorIntersectionsToNodes(FAPio.wr...
...iteCenters);
        }
    }

    // Thread class to create nodes representing Stars
    private static class addStarsToNodes2 implements Runnable{
        public void run(){
            NetworkBuilder.addStarsToNodes();
        }
    }

    // Thread to create links within the sector
    private static class addIntraSectorLinksFromNodes2 implements Runna...
...ble{
        public void run(){
            NetworkBuilder.addIntraSectorLinksFromNodes();
        }
    }

    // Thread to create Victor Airway links
    private static class addVictorAirwayNetworkLinks2 implements Runnab...
...le{
        public void run(){
            NetworkBuilder.addVictorAirwayNetworkLinks();
        }
    }

    // Thread to create Jet Route links
    private static class addJetRouteNetworkLinks2 implements Runnable{
        public void run(){
            NetworkBuilder.addJetRouteNetworkLinks();
        }
    }

    // Thread to creat RNAV route links
    private static class addRnavRouteLinks2 implements Runnable{
        public void run(){
```

```
            NetworkBuilder.addRnavRouteLinks();
        }
    }

    public static void buildNetwork(){
        Airspace a = APData.airspace;
        Network n = APData.network;



        Thread tNavaids = new Thread(new addNavaidsAndFixesToNodes2());...
...        tNavaids.start();

        try{
            //tStars.join();
            tNavaids.join();
        }catch(InterruptedException e){
            System.out.println ("Error creating network nodes.");
        }

        Thread tAirports = new Thread(new addAirportsToNodes2());
        tAirports.start();



        Thread tInter = new Thread(new addAirwaySectorIntersectionsToNo...
...des2());
        Thread tStars = new Thread(new addStarsToNodes2());

        // Wait for airports to stop before starting navaids
        try{
            tAirports.join();
            tInter.start();
        }catch(InterruptedException e){
            System.out.println("Error creating network nodes.");
        }

        // Wait for inter to stop before starting stars
        try{
            tInter.join();
            tStars.start();
        }catch(InterruptedException e){
            System.out.println("Error creating network nodes.");
        }


        // Nodes must finish before links are created
```

```
        try{
            tStars.join();
            //tNavaids.join();
        }catch(InterruptedException e){
            System.out.println ("Error creating network nodes.");
        }

        Thread tIntra = new Thread(new addIntraSectorLinksFromNodes2())...
...;
        tIntra.start();

        try{
            tIntra.join();
            //tVictor.join();
            //tJet.join();
            //tRnav.join();
        }catch(InterruptedException e){
            System.out.println("Error creating network links.");
        }



        Thread tJet = new Thread(new addJetRouteNetworkLinks2());
        tJet.start();

        try{
            //tIntra.join();
            //tVictor.join();
            tJet.join();
            //tRnav.join();
        }catch(InterruptedException e){
            System.out.println("Error creating network links.");
        }

        Thread tRnav = new Thread(new addRnavRouteLinks2());
        tRnav.start();
        try{
            //tIntra.join();
            //tVictor.join();
            //tJet.join();
            tRnav.join();
        }catch(InterruptedException e){
            System.out.println("Error creating network links.");
        }

        Thread tVictor = new Thread(new addVictorAirwayNetworkLinks2())...
...;
```

```
        tVictor.start();

        try{
            //tIntra.join();
            tVictor.join();
            //tJet.join();
            //tRnav.join();
        }catch(InterruptedException e){
            System.out.println("Error creating network links.");
        }

        // Airport links adds Nodes
        NetworkBuilder.addAirportLinks(true);
        NetworkBuilder.addAirportLinks(false);

        // Flight Plan links must be added last
        NetworkBuilder.addAdditionalLinksFromFlightPlan();

        n.setFloorLowestLinksToGround();
        //n.trimArrayLists();
    }

    // Identifiers such as SC32_CEILING_350
    public static int getAltitudeFromNodeIdentifier(String identifier){...
...         Node3D<Double> idNode = APData.network.getNode(identifier);
        if(idNode.getParentSector().equals(ASector.NullSector)){
            return (int)DEFAULTNODEELEVATION;
        }

        int beginIndexAltitude = identifier.lastIndexOf("_") + 1;

        String altitudeString = identifier.substring(beginIndexAltitude...
...);
        int altitude = Integer.parseInt(altitudeString);

        return altitude;
    }

    // Add standard terminal arrivals to nodes
    public static void addStarsToNodes(){
        Airspace a = APData.airspace;
        Network n = APData.network;

        Iterator<String> e = a.getStarIterator();
        while(e.hasNext()){
            String starId = e.next();
            GPoint2D<Double> starPt = a.getStar(starId);
```

```
            Node3D<Double> starNode = new Node3D<Double>(
                starPt.getLongitude(), starPt.getLatitude(), DEFAULTFIR...
...STNODEELEV,
                Node3D.NodeTypes.Star, starId);
            starNode.setParentSector(ASector.NullSector);
            n.addNode(starId, starNode);
        }
    }

    // Add additional links contained in the flight plans but not inclu...
...ded in
    // the network thus far.
    public static void addAdditionalLinksFromFlightPlan(){
        Airspace a = APData.airspace;
        Network n = APData.network;

        int totalLinks = 0;
        int additionalLinks = 0;

        // Iterate through each of the flights
        Iterator<String> e = a.getFlightIterator();
        while(e.hasNext()){
            Flight f = a.getFlight(e.next());
            FTrajectory t = f.getPlannedTrajectory();

            totalLinks += t.noWaypoints() - 1;

            for(int i=1; i<t.noWaypoints(); i++){
                String firstId = t.getWaypointIdentifier(i-1);
                String secondId = t.getWaypointIdentifier(i);

                // Case where firstId and/or secondId are also airports...
...                String candidateLink = firstId + APData.BAR + secondId;...
...                boolean needToAdd = n.hasNode(firstId) && n.hasNode(sec...
...ondId)
                    && !n.hasLink(candidateLink);


                if(!n.hasLink(firstId, secondId) || needToAdd){
                    additionalLinks++;
                    // Add the link
                    ArrayList<String> firstIds = n.getNodesUsingIdentif...
...ier(firstId);
                    ArrayList<String> secondIds = n.getNodesUsingIdenti...
...fier(secondId);
```

```
                    for(int k=0; k<firstIds.size(); k++){
                        String node1S = firstIds.get(k);
                        Node3D<Double> node1 = n.getNode(node1S);

                        boolean node1IsExternal = node1.getParentSector...
...().equals(ASector.NullSector);

                        // Only consider centroid and external nodes
                        if(node1.isAtBoundary() || node1.isAtFloor() ||...
...                                 node1.isAtCeiling() || node1.isAtAirpor...
...t()) continue;

                        // Only add one link that most closely matches ...
...the
                        // altitude limits of the first node's sector
                        double minAltitudeDiff = 9999.0D;
                        double floor = 0.0D;
                        double ceiling = 0.0D;
                        String minDiffNode2S = null;

                        ASector sec1 = node1.getParentSector();
                        for(int j=0; j<secondIds.size(); j++){
                            String node2S = secondIds.get(j);
                            Node3D<Double> node2 = n.getNode(node2S);

                            // Only consider centroid and external node...
...s
                            if(node2.isAtBoundary() || node2.isAtFloor(...
...) ||
                                    node2.isAtCeiling() || node2.isAtAi...
...rport()) continue;

                            ASector sec2 = node2.getParentSector();

                            double floorAltDiff = Math.abs(sec1.getFloo...
...r() - sec2.getFloor());
                            double ceilingAltDiff = Math.abs(sec1.getCe...
...iling() - sec2.getCeiling());
                            double altDiff = floorAltDiff + ceilingAltD...
...iff;

                            // Found a node with parent sector that has...
... altitude
                            // limits that match more closely.
                            if(altDiff < minAltitudeDiff){
                                minAltitudeDiff = altDiff;
```

```
                                if(node1IsExternal){
                                    floor = sec2.getFloor();
                                }else{
                                    floor = Math.min(sec1.getFloor(), s...
...ec2.getFloor());
                                }

                                ceiling = Math.max(sec1.getCeiling(), s...
...ec2.getCeiling());
                                minDiffNode2S = node2S;
                            }

                            //n.addLink(node1S,node2S,floor,ceiling, Li...
...nk3D.LinkTypes.AdditionalLinks);
                        }

                        // Check to see if there is a link from the cen...
...troid to the
                        // boundary.
                        Node3D<Double> firstNode = n.getNode(node1S);
                        ArrayList<Link3D> firstNodeOutlinks = firstNode...
....getOutLinkList();
                        boolean linkToBoundaryFound1 = false;

                        Node3D<Double> secondNode = n.getNode(minDiffNo...
...de2S);
                        ArrayList<Link3D> secondNodeOutlinks = secondNo...
...de.getOutLinkList();
                        boolean linkToBoundaryFound2 = false;

                        for(Link3D outLink:firstNodeOutlinks){
                            if(outLink.getToNode().isAtBoundary() &&
                                    outLink.getIdentifier().contains(se...
...condNode.getAirspaceIdentifier())){
                                    linkToBoundaryFound1 = true;
                                    break;
                            }
                        }
                        for(Link3D outLink:secondNodeOutlinks){
                            if(outLink.getToNode().isAtBoundary() &&
                                    outLink.getIdentifier().contains(fi...
...rstNode.getAirspaceIdentifier())){
                                    linkToBoundaryFound2 = true;
                                    break;
                            }
                        }
                        // If links to boundary not found then add the ...
```

```
...link
                        if(!linkToBoundaryFound1 || !linkToBoundaryFoun...
...d2){

                            n.addLink(node1S, minDiffNode2S, floor, cei...
...ling,
                                Link3D.LinkTypes.AdditionalLinks);
                        }
                    }
                }
            }
        }

        System.out.println("Total number of flight plan links: " + Inte...
...ger.toString(totalLinks));
        System.out.println("Additional links required: " + Integer.toSt...
...ring(additionalLinks));
    }

    // Add links between airport and the first navaid/fix on the route
    public static void addAirportLinks(boolean addOriginAirportLinks){
        Airspace a = APData.airspace;
        Network n = APData.network;

        Iterator<String> e = a.getFlightIterator();

        int unidentified = 0;

        while(e.hasNext()){
            String flightId = e.next();

            Flight f = a.getFlight(flightId);

            String airportId = addOriginAirportLinks ?
                f.getOriginAirport().getIdentifier() + APData._APT :
                f.getDestinationAirport().getIdentifier() + APData._APT...
...;

            FTrajectory t = f.getPlannedTrajectory();

            // Get first waypoint in the route
            String firstPoint = addOriginAirportLinks ?
                t.getWaypointIdentifier(0) :
                t.getWaypointIdentifier(t.noWaypoints()-1);

            // Check if any nodes are using the identifier
            if(!n.hasNode(firstPoint)){
```

```
                unidentified++;

                GPoint4D<Double> pt = addOriginAirportLinks ?
                    t.getWaypoint(0) :
                    t.getWaypoint(t.noWaypoints()-1);

                // Add the point as a node
                Node3D<Double> node = new Node3D<Double>(pt.getLongitud...
...e(),
                        pt.getLatitude(), DEFAULTFIRSTNODEELEV,
                        Node3D.NodeTypes.FixCentroid, firstPoint);
                node.setParentSector(ASector.NullSector);
                n.addNode(firstPoint, node);

            // If there is only one node using the identifier then a se...
...arch
            // is not necessary.
            }else{
                ArrayList<String> firstPtNodes = n.getNodesUsingIdentif...
...ier(firstPoint);

                // Get the lowest centroid or STAR node
                int minAltitude = 999;

                for(int i=0; i<firstPtNodes.size(); i++){
                    String nodeFirstId = firstPtNodes.get(i);
                    Node3D<Double> node = n.getNode(nodeFirstId);

                    // Check that the last character is a digit
                    if(node.isAtAirport()){
                        continue;
                    }else if(node.isAtStar() || node.getParentSector()....
...equals(ASector.NullSector)){
                        firstPoint = nodeFirstId;
                        break;
                    }else if(!node.isAtCentroid()){
                        continue;
                    }

                    int lastUnderIdx = nodeFirstId.lastIndexOf(APData.U...
...NDERSCORE);
                    //String[] tmpStr = nodeFirstId.split(APData.UNDERS...
...CORE);
                    int nodeAltitude = Integer.parseInt(
                            nodeFirstId.substring(lastUnderIdx+1,nodeFi...
...rstId.length()));
                    if(nodeAltitude < minAltitude){
```

```
                    minAltitude = nodeAltitude;
                    firstPoint = nodeFirstId;
                }
            }

        }

        // Get the node object to establish ceiling elevation for t...
...he link
        Node3D<Double> nodePt = n.getNode(firstPoint);
        ASector nodePtSector = nodePt.getParentSector();
        double ceiling = 999.0D;
        if(!nodePtSector.equals(ASector.NullSector)){
            ceiling = nodePtSector.hasFix(nodePt.getAirspaceIdentif...
...ier()) ?
                nodePtSector.getFixModuleCeiling(nodePt.getAirspace...
...Identifier()) :
                nodePtSector.getNavaidModuleCeiling(nodePt.getAirsp...
...aceIdentifier());
        }

        // Create the link
        n.addLink(airportId, firstPoint, 0D, ceiling, Link3D.LinkTy...
...pes.AirportConnector);
        n.addLink(firstPoint, airportId, 0D, ceiling, Link3D.LinkTy...
...pes.AirportConnector);
    }

    System.out.println("Number of nodes to be added at first identi...
...fier: "
            + Integer.toString(unidentified));
}

// Add links between Navaids and Fixes on an Airway
public static void addVictorAirwayNetworkLinks(){
    Airspace a = APData.airspace;
    Network n = APData.network;

    Iterator<String> e = a.getVictorAirwayIterator();
    while(e.hasNext()){
        String airwayIdentifier = e.next();

        Airway awy = a.getVictorAirway(airwayIdentifier);

        addAirwayLinks(awy, Link3D.LinkTypes.VictorAirway);
    }
}
```

```
    public static void addJetRouteNetworkLinks(){
        Airspace a = APData.airspace;
        Network n = APData.network;

        Iterator<String> e = a.getJetRouteIterator();
        while(e.hasNext()){
            String jetRouteIdentifier = e.next();

            Airway awy = a.getJetRoute(jetRouteIdentifier);

            addAirwayLinks(awy, Link3D.LinkTypes.JetRoute);
        }
    }

    public static void addRnavRouteLinks(){
        Airspace a = APData.airspace;
        Network n = APData.network;

        Iterator<String> e = a.getRnavRouteIterator();
        while(e.hasNext()){
            String rnavRouteIdentifier = e.next();

            Airway awy = a.getRnavRoute(rnavRouteIdentifier);

            addAirwayLinks(awy, Link3D.LinkTypes.RnavRoute);

        }
    }

    public static void addAirwayLinks(Airway awy, Link3D.LinkTypes link...
...Type){
        Network n = APData.network;

        for(int i=1; i<awy.noPoints(); i++){
            String firstId = awy.getPointIdentifier(i-1);
            String secondId = awy.getPointIdentifier(i);
            ArrayList<String> firstNodes = n.getNodesUsingIdentifier(fi...
...rstId);
            ArrayList<String> secondNodes = n.getNodesUsingIdentifier(s...
...econdId);

            for(int j=0; j<firstNodes.size(); j++){
                String firstNodeId = firstNodes.get(j);
                Node3D<Double> firstNode = n.getNode(firstNodeId);
                ASector firstSector = firstNode.getParentSector();
                boolean firstNodeIsBoundary = false;
```

```
                boolean firstNodeIsExternal = false;

                // Do not consider floor and ceiling nodes
                if(firstNode.isAtFloor() || firstNode.isAtCeiling()){
                    continue;
                }else if(firstNode.isAtBoundary()){
                    firstNodeIsBoundary = true;
                }else if(firstSector.equals(ASector.NullSector)){
                    firstNodeIsExternal = true;
                }

                for(int k=0; k<secondNodes.size(); k++){
                    String secondNodeId = secondNodes.get(k);

                    Node3D<Double> secondNode = n.getNode(secondNodeId)...
...;
                    ASector secondSector = secondNode.getParentSector()...
...;
                    boolean secondNodeIsExternal = secondSector.equals(...
...ASector.NullSector);

                    // Get the altitude limits for the link
                    double linkFloor = Math.min(firstSector.getFloor(),...
... secondSector.getFloor());
                    double linkCeiling = Math.max(firstSector.getCeilin...
...g(), secondSector.getCeiling());
//                    double linkFloor = 999.0D;
//                    double linkCeiling = 0.0D;
//                    if(firstSector.hasFix(firstId)){
//                        linkFloor = Math.min(firstSector.getFixModule...
...Floor(firstId),linkFloor);
//                        linkCeiling = Math.max(firstSector.getFixModu...
...leCeiling(firstId),linkCeiling);
//                    }else if(firstSector.hasNavaid(firstId)){
//                        linkFloor = Math.min(firstSector.getNavaidMod...
...uleFloor(firstId),linkFloor);
//                        linkCeiling = Math.max(firstSector.getNavaidM...
...oduleCeiling(firstId),linkCeiling);
//                    }
//
//                    if(secondSector.hasFix(secondId)){
//                        linkFloor = Math.min(secondSector.getFixModul...
...eFloor(secondId),linkFloor);
//                        linkCeiling = Math.max(secondSector.getFixMod...
...uleCeiling(secondId),linkCeiling);
//                    }else if(secondSector.hasNavaid(secondId)){
//                        linkFloor = Math.min(secondSector.getNavaidMo...
```

```
...duleFloor(secondId),linkFloor);
//                        linkCeiling = Math.max(secondSector.getNavaid...
...ModuleCeiling(secondId),linkCeiling);
//                    }


                    // Only consider centroid nodes
                    if(secondNode.isAtFloor() || secondNode.isAtCeiling...
...()){
                        continue;
                    }else if(firstNodeIsBoundary && secondNode.isAtBoun...
...dary() &&
                            firstNodeId.contains(secondId) && secondNod...
...eId.contains(firstId)){
                        // Add links between sectors
                        n.addLink(firstNodeId, secondNodeId, linkFloor,...
... linkCeiling, linkType);
                        n.addLink(secondNodeId, firstNodeId, linkFloor,...
... linkCeiling, linkType);

                        continue;
                    }else if((firstNodeIsExternal && secondNode.isAtBou...
...ndary()) ||
                            firstNodeIsBoundary && secondNodeIsExternal...
...){
                        // Add links between an external node and a sec...
...tor
                        n.addLink(firstNodeId, secondNodeId, linkFloor,...
... linkCeiling, linkType);
                        n.addLink(secondNodeId, firstNodeId, linkFloor,...
... linkCeiling, linkType);
                    }else if(firstNodeIsBoundary || secondNode.isAtBoun...
...dary()){
                        continue;
                    }

                    // Both nodes are in the same sector
                    if(firstSector.equals(secondSector)){
                        n.addLink(firstNodeId, secondNodeId, linkFloor,...
... linkCeiling, linkType);
                        n.addLink(secondNodeId, firstNodeId, linkFloor,...
... linkCeiling, linkType);
                    }
                }
            }

        }
```

```
    }

    // Create links from nodes within the sector having the same identi...
...fier
    public static void addIntraSectorLinksFromNodes(){
        Network n = APData.network;

        int boundaryLinksCount = 0;

        // Iterate through identifiers (e.g. NAVAIDs, Fixes) used to cr...
...eate
        // the nodes
        Iterator<String> e = n.getIdentifierIterator();
        while(e.hasNext()){
            String identifier = e.next();

            // Get the list of flight plan nodes using the identifier
            ArrayList<String> nodeIdentifiers = n.getNodesUsingIdentifi...
...er(identifier);

            // Classify nodes by type
            ArrayList<String> nodeFloor = new ArrayList<String>();
            ArrayList<String> nodeCentroid = new ArrayList<String>();
            ArrayList<String> nodeCeiling = new ArrayList<String>();
            ArrayList<String> nodeBoundary = new ArrayList<String>();

            // Add node identifiers to types
            for(String nodeId:nodeIdentifiers){
                Node3D<Double> node = n.getNode(nodeId);
                if(node.isAtFloor()){
                    nodeFloor.add(nodeId);
                }else if(node.isAtCentroid()){
                    nodeCentroid.add(nodeId);
                }else if(node.isAtCeiling()){
                    nodeCeiling.add(nodeId);
                }else if(node.isAtBoundary()){
                    nodeBoundary.add(nodeId);
                }
            }

            // Flag for the first iteration
            boolean firstIteration = true;

            // Build the links from the floor up
            int nodesRemaining = 1;
            while(nodesRemaining > 0){
                // First create links attached to the sector module flo...
```

```
...or
                if(nodeFloor.size() > 0){
                    // Find the lowest node floor
                    int minFloor = 9999;
                    String minFloorId = null;
                    for(String nodeId:nodeFloor){
                        int altitude = NetworkBuilder.getAltitudeFromNo...
...deIdentifier(nodeId);
                        if(altitude < minFloor){
                            minFloor = altitude;
                            minFloorId = nodeId;
                        }
                    }

                    // Remove the floor node from further consideration...
...                    nodeFloor.remove(minFloorId);

                    // Find the highest ceiling less than the node floo...
...r
                    int maxCeiling = 0;
                    String maxCeilingId = null;
                    boolean ceilingExists = false;
                    for(String nodeId:nodeCeiling){
                        int altitude = NetworkBuilder.getAltitudeFromNo...
...deIdentifier(nodeId);
                        if(altitude > maxCeiling && altitude <= minFloo...
...r){
                            maxCeiling = altitude;
                            maxCeilingId = nodeId;
                            ceilingExists = true;
                        }
                    }

                    // Get the altitude limits
                    Node3D<Double> node1 = n.getNode(minFloorId);
                    ASector sec1 = node1.getParentSector();
                    double floor = sec1.getFloor();
                    double ceiling = sec1.getCeiling();
                    if(sec1.hasFix(identifier)){
                        floor = sec1.getFixModuleFloor(identifier);
                        ceiling = sec1.getFixModuleCeiling(identifier);...
...                    }else if(sec1.hasNavaid(identifier)){
                        floor = sec1.getNavaidModuleFloor(identifier);
                        ceiling = sec1.getNavaidModuleCeiling(identifie...
...r);
                    }
```

```
                    // If the ceiling exists create the links
                    if(ceilingExists){
                        n.addLink(minFloorId, maxCeilingId, floor, ceil...
...ing,
                                Link3D.LinkTypes.Intrasector);
                        n.addLink(maxCeilingId, minFloorId, floor, ceil...
...ing,
                                Link3D.LinkTypes.Intrasector);
                    }

                    // Find the lowest centroid that is greater than th...
...e floor
                    int minCentroid = 9999;
                    String minCentroidId = null;
                    for(String nodeId:nodeCentroid){
                        int altitude = NetworkBuilder.getAltitudeFromNo...
...deIdentifier(nodeId);
                        if(altitude < minCentroid && altitude > minFloo...
...r){
                            minCentroid = altitude;
                            minCentroidId = nodeId;
                        }
                    }

                    // Remove the centroid node from further considerat...
...ion
                    nodeCentroid.remove(minCentroidId);

                    // Add the centroid to floor links
                    n.addLink(minFloorId, minCentroidId, floor, ceiling...
...,
                            Link3D.LinkTypes.Intrasector);
                    n.addLink(minCentroidId, minFloorId, floor, ceiling...
...,
                            Link3D.LinkTypes.Intrasector);

                    // Find the ceiling above the centroid
                    int minCeiling = 9999;
                    String minCeilingId = null;
                    for(String nodeId:nodeCeiling){
                        int altitude = NetworkBuilder.getAltitudeFromNo...
...deIdentifier(nodeId);
                        if(altitude < minCeiling && altitude > minCentr...
...oid){
                            minCeiling = altitude;
                            minCeilingId = nodeId;
                        }
```

```
                    }

                    // Add links between centroid and ceiling
                    n.addLink(minCentroidId, minCeilingId, floor, ceili...
...ng,
                            Link3D.LinkTypes.Intrasector);
                    n.addLink(minCeilingId, minCentroidId, floor, ceili...
...ng,
                            Link3D.LinkTypes.Intrasector);

                    // Find all boundary nodes between the floor and th...
...e ceiling
                    if(firstIteration){
                        minFloor = 0;
                        firstIteration = false;
                    }
                    for(String nodeId:nodeBoundary){
                        //int altitude = NetworkBuilder.getAltitudeFrom...
...NodeIdentifier(nodeId);
                        //if(altitude > minFloor && altitude < minCeili...
...ng){
                        Node3D<Double> boundaryNode = n.getNode(nodeId)...
...;
                        Node3D<Double> centroidNode = n.getNode(minCent...
...roidId);
                        ASector boundarySector = boundaryNode.getParent...
...Sector();
                        ASector centroidSector = centroidNode.getParent...
...Sector();
                        boolean sameFloor = Math.abs(boundarySector.get...
...Floor() -
                                centroidSector.getFloor()) <= 1.0;
                        boolean sameCeiling = Math.abs(boundarySector.g...
...etCeiling() -
                                centroidSector.getCeiling()) <= 1.0;
                        if(sameFloor && sameCeiling){
                            boundaryNode.setParentSector(centroidSector...
...);
                            boundarySector = centroidSector;
                        }

                        if(boundarySector.equals(centroidSector)){
                            boundaryLinksCount++;
                            n.addLink(minCentroidId, nodeId, floor, cei...
...ling,
                                    Link3D.LinkTypes.Intrasector);
                            n.addLink(nodeId, minCentroidId, floor, cei...
```

```
...ling,
                                    Link3D.LinkTypes.Intrasector);

                        }
                    }
                }

                nodesRemaining = nodeFloor.size();
            }

            nodeFloor = null;
            nodeCentroid = null;
            nodeCeiling = null;
            nodeBoundary = null;

        }

        System.out.println("*****Number of boundary links added: " +
                Integer.toString(boundaryLinksCount));

    }

    public static void addAirportsToNodes(Iterator<String> flightIterat...
...or){
        while(flightIterator.hasNext()){
            String callSign = flightIterator.next();

            // Only consider airports that are used by IFR ETMS flight ...
...plans
            Flight f = APData.airspace.getFlight(callSign);

            // Airport nodes
            Airport originAirport = f.getOriginAirport();
            Airport destinationAirport = f.getDestinationAirport();
            String aptClass = originAirport.getClass().getName();

            // Set sector for these airports to null if not already set...
...             originAirport.setParentSector(ASector.NullSector);
            destinationAirport.setParentSector(ASector.NullSector);

            // Need to add a suffix to the airport identifier to distin...
...guish
            // from Navaid identifiers.
            String originAptId = originAirport.getIdentifier() + APData...
...._APT;
            String destinationAptId = destinationAirport.getIdentifier(...
...) + APData._APT;
```

```
            APData.network.addNode(originAptId,originAirport);
            APData.network.addNode(destinationAptId,destinationAirport)...
...;
        }

    }

    public static void addAirwaySectorIntersectionsToNodes(String[] cen...
...tersToWrite){
        Airspace a = APData.airspace;

        int boundaryNodesCount = 0;

        Iterator<ACenter> centerIterator = a.getCenterValueIterator();
        while(centerIterator.hasNext()){
            ACenter center = centerIterator.next();

            Iterator<ASector> sectorIterator = center.getSectorValueIte...
...rator();
            while(sectorIterator.hasNext()){
                ASector sector = sectorIterator.next();

                Iterator<String> intersectionIterator = sector.getAirwa...
...yIntersectionIterator();
                while(intersectionIterator.hasNext()){
                    String fromId = intersectionIterator.next();

                    Iterator<String> ptS = sector.getToWaypointIntersec...
...tionIterator(fromId);
                    while(ptS.hasNext()){
                        String toId = ptS.next();

                        GPoint3D<Double> ptG = sector.getAirwayIntersec...
...tionLocation(
                                fromId, toId);

                        // Create a new node using the coordinates of t...
...he airway
                        // intersection.
                        Node3D<Double> pt = new Node3D<Double>(ptG, Nod...
...e3D.NodeTypes.Boundary,
                                fromId);
                        pt.setParentSector(sector);

                        long elevation = Math.round(pt.getElevation());...
...
```

```
                        String nodeId = fromId + APData.UNDERSCORE + to...
...Id + APData._BOUNDARY_ +
                                Long.toString(elevation);

                        APData.network.addNode(nodeId, pt);

                        boundaryNodesCount++;
                    }
                }
            }
        }

        System.out.println("*****Number of boundary nodes added: " +
                Integer.toString(boundaryNodesCount));
    }

    public static void addNavaidsAndFixesToNodes(String[] centersToWrit...
...e){
        Airspace a = APData.airspace;
        Network n = APData.network;
        for(String centerS:centersToWrite){
            // Center element
            ACenter center = a.getCenter(centerS);

            // Iterate through sectors
            Iterator<ASector> sectorIterator = center.getSectorValueIte...
...rator();
            while(sectorIterator.hasNext()){
                ASector sector = sectorIterator.next();

                // Iterate through NAVAIDs
                Iterator<String> navaidIterator = sector.getNavaidsInSe...
...ctorIterator();
                while(navaidIterator.hasNext()){
                    String navaidId = navaidIterator.next();

                    if(!a.hasNavaid(navaidId)) continue;

                    // Get altitude of floor and ceiling to calculate c...
...entroid
                    double sectorFloor = sector.getNavaidModuleFloor(na...
...vaidId);
                    double sectorCeiling = sector.getNavaidModuleCeilin...
...g(navaidId);

                    // Check to see if within limits
                    if(sectorFloor < MINSECTORFLOORALT || sectorCeiling...
```

```
... < MINSECTORCEILALT){
                        continue;
                    }

                    // Calculate vertical centroid of sector
                    double sectorCentroid = (sectorFloor + sectorCeilin...
...g)*0.5D;

                    // Round and convert to long
                    long sectorFloorL = Math.round(sectorFloor);
                    long sectorCeilingL = Math.round(sectorCeiling);
                    long sectorCentroidL = Math.round(sectorCentroid);

                    // Get lat/lon
                    GPoint2D<Double> navaid = a.getNavaid(navaidId);

                    // Create node at sector floor
                    String navaidIdFloor = navaidId + APData._FLOOR_ +
                            Long.toString(sectorFloorL);
                    Node3D<Double> nodeFloor = new Node3D<Double>(
                            navaid.getLongitude(), navaid.getLatitude()...
..., sectorFloor,
                            Node3D.NodeTypes.NavaidFloor, navaidId);
                    nodeFloor.setParentSector(sector);
                    n.addNode(navaidIdFloor, nodeFloor);

                    // Create node at sector ceiling
                    String navaidIdCeiling = navaidId + APData._CEILING...
..._ +
                            Long.toString(sectorCeilingL);
                    Node3D<Double> nodeCeiling = new Node3D<Double>(
                            navaid.getLongitude(), navaid.getLatitude()...
..., sectorCeiling,
                            Node3D.NodeTypes.NavaidCeiling, navaidId);
                    nodeCeiling.setParentSector(sector);
                    n.addNode(navaidIdCeiling, nodeCeiling);

                    // Create node at sector vertical centroid
                    String navaidIdCentroid = navaidId + APData._CENTRO...
...ID_ +
                            Long.toString(sectorCentroidL);
                    Node3D<Double> nodeCentroid = new Node3D<Double>(
                            navaid.getLongitude(), navaid.getLatitude()...
..., sectorCentroid,
                            Node3D.NodeTypes.NavaidCentroid, navaidId);...
...                    nodeCentroid.setParentSector(sector);
                    n.addNode(navaidIdCentroid, nodeCentroid);
```

```
                }


                // Do the same thing for Fixes
                Iterator<String> fixIterator = sector.getFixesInSectorI...
...terator();
                while(fixIterator.hasNext()){
                    String fixId = fixIterator.next();

                    if(!a.hasFix(fixId)) continue;

                    // Get altitude of floor and ceiling to calculate c...
...entroid
                    double sectorFloor = sector.getFixModuleFloor(fixId...
...);
                    double sectorCeiling = sector.getFixModuleCeiling(f...
...ixId);

                    // Check to see if within limits
                    if(sectorFloor < MINSECTORFLOORALT || sectorCeiling...
... < MINSECTORCEILALT){
                        continue;
                    }

                    // Calculate vertical centroid of sector
                    double sectorCentroid = (sectorFloor + sectorCeilin...
...g)*0.5D;

                    // Round and convert to long
                    long sectorFloorL = Math.round(sectorFloor);
                    long sectorCeilingL = Math.round(sectorCeiling);
                    long sectorCentroidL = Math.round(sectorCentroid);

                    // Get lat/lon
                    GPoint2D<Double> fix = a.getFix(fixId);

                    // Create node at sector floor
                    String fixIdFloor = fixId + APData._FLOOR_ +
                            Long.toString(sectorFloorL);
                    Node3D<Double> nodeFloor = new Node3D<Double>(
                            fix.getLongitude(), fix.getLatitude(), sect...
...orFloor,
                            Node3D.NodeTypes.FixFloor, fixId);
                    nodeFloor.setParentSector(sector);
                    n.addNode(fixIdFloor, nodeFloor);

                    // Create node at sector ceiling
```

```
                    String fixIdCeiling = fixId + APData._CEILING_ +
                            Long.toString(sectorCeilingL);
                    Node3D<Double> nodeCeiling = new Node3D<Double>(
                            fix.getLongitude(), fix.getLatitude(), sect...
...orCeiling,
                            Node3D.NodeTypes.FixCeiling, fixId);
                    nodeCeiling.setParentSector(sector);
                    n.addNode(fixIdCeiling, nodeCeiling);

                    // Create node at sector vertical centroid
                    String fixIdCentroid = fixId + APData._CENTROID_ +
                            Long.toString(sectorCentroidL);
                    Node3D<Double> nodeCentroid = new Node3D<Double>(
                            fix.getLongitude(), fix.getLatitude(), sect...
...orCentroid,
                            Node3D.NodeTypes.FixCentroid, fixId);
                    nodeCentroid.setParentSector(sector);
                    n.addNode(fixIdCentroid, nodeCentroid);
                }
            }
        }

        // Add Navaids and Fixes outside of the centers and sectors und...
...er analysis
        Iterator<String> e = a.getFixIterator();
        while(e.hasNext()){
            String fixId = e.next();

            // Do not consider fix if it is already defined in a sector...
...            if(n.hasNode(fixId)){
                int maxAltitude = 0;
                ArrayList<String> fixIds = new ArrayList<String>(n.getN...
...odesUsingIdentifier(fixId));
                for(String tmpFixId:fixIds){
                    int tmpMaxAltitude = getAltitudeFromNodeIdentifier(...
...tmpFixId);
                    if(tmpMaxAltitude > maxAltitude) maxAltitude = tmpM...
...axAltitude;
                }

                // Check max altitude
                if(maxAltitude == DEFAULTNODEELEVATION || maxAltitude >...
... 900){
                    continue;
                }else{
                    for(String tmpFixId:fixIds){
                        n.removeNode(tmpFixId);
```

```
                }
            }
        }

        GPoint2D<Double> pt = a.getFix(fixId);

        Node3D<Double> node = new Node3D<Double>(pt.getLongitude(),...
...                 pt.getLatitude(), DEFAULTNODEELEVATION,
                Node3D.NodeTypes.FixCentroid, fixId);
        node.setParentSector(ASector.NullSector);

        n.addNode(fixId, node);

    }

    e = a.getNavaidIterator();
    while(e.hasNext()){
        String navaidId = e.next();

        if(n.hasNode(navaidId)){
            int maxAltitude = 0;
            ArrayList<String> navaidIds = new ArrayList<String>(n.g...
...etNodesUsingIdentifier(navaidId));

            // Only one node - added as a fix previously
            if(navaidIds.size() == 1) continue;

            for(String tmpNavaidId:navaidIds){
                int tmpMaxAltitude = getAltitudeFromNodeIdentifier(...
...tmpNavaidId);
                if(tmpMaxAltitude > maxAltitude) maxAltitude = tmpM...
...axAltitude;
            }

            // Check max altitude
            if(maxAltitude == DEFAULTNODEELEVATION || maxAltitude >...
... 900){
                continue;
            }else{
                for(String tmpNavaidId:navaidIds){
                    n.removeNode(tmpNavaidId);
                }
            }

            //continue;
        }
```

```
            GPoint2D<Double> pt = a.getNavaid(navaidId);

            Node3D<Double> node = new Node3D<Double>(pt.getLongitude(),...
...                     pt.getLatitude(), DEFAULTNODEELEVATION,
                    Node3D.NodeTypes.NavaidCentroid, navaidId);
            node.setParentSector(ASector.NullSector);

            n.addNode(navaidId, node);

        }
    }
}
```

Node3D.java

```
/*
 * Node3D.java
 *
 * Created on May 25, 2007, 11:38 AM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author jeff
 */
import java.util.*;
public class Node3D<E extends Number> extends GPoint3D<E> {
    private static final int NUMOUTLINKS = 5;

    // Define what type of node this is.
    public enum NodeTypes {
        Airport, NavaidFloor, NavaidCentroid, NavaidCeiling,
            FixFloor, FixCentroid, FixCeiling, Star, Boundary, Copy
    }

    //private long nodeID
    private final NodeTypes nodeType_;
    private ArrayList<Link3D> outLinkList_;

    // Parent sector (where this node is located)
    private ASector parentSector_;


    // Identifier of the original airspace identifier
    private final String airspaceIdentifier_;

    private double[] doubleValues_;
    private int[] integerValues_;

    /** Creates a new instance of Node3D */
    public Node3D(E longitude, E latitude, E elevation,
            NodeTypes nodeType, String airspaceIdentifier){
        super.setLongitude(longitude);
        super.setLatitude(latitude);
        super.setElevation(elevation);
        parentSector_ = ASector.NullSector;
        outLinkList_ = new ArrayList<Link3D>(NUMOUTLINKS);
        nodeType_ = nodeType;
```

```
        airspaceIdentifier_ = airspaceIdentifier;
    }

    public Node3D(GPoint3D<E> pt, NodeTypes nodeType,
            String airspaceIdentifier){
        super(pt);
        parentSector_ = ASector.NullSector;
        outLinkList_ = new ArrayList<Link3D>();
        nodeType_ = nodeType;
        airspaceIdentifier_ = airspaceIdentifier;
    }

    public Node3D(){
        outLinkList_ = new ArrayList<Link3D>();
        nodeType_ = NodeTypes.Copy;
        airspaceIdentifier_ = APData.NULLSTRING;
    }

    public Node3D<E> initializeDoubleValues(int numValues){
        doubleValues_ = new double[numValues];
        for(int i=0; i<doubleValues_.length; i++) doubleValues_[i] = 0....
...0D;
        return this;
    }

    public Node3D<E> setDoubleValue(int index, double val){
        doubleValues_[index] = val;
        return this;
    }

    public double getDoubleValue(int index){
        if(doubleValues_ == null || index >= doubleValues_.length)
            return Double.NaN;
        return doubleValues_[index];
    }

    public boolean hasDoubleValueAtIndex(int index){
        if(doubleValues_ == null) return false;
        if(doubleValues_.length <= index) return false;

        return true;
    }

    public Node3D<E> initializeIntegerValues(int numValues){
        integerValues_ = new int[numValues];
        for(int i=0; i<doubleValues_.length; i++) doubleValues_[i] = 0;...
...        return this;
```

```
    }

    public Node3D<E> setIntegerValue(int index, int val){
        integerValues_[index] = val;
        return this;
    }

    public int getIntegerValue(int index){
        return integerValues_[index];
    }

    public String getAirspaceIdentifier(){
        return airspaceIdentifier_;
    }

    public boolean hasSameAirspaceIdentifier(Node3D node){
        return airspaceIdentifier_.equals(node.getAirspaceIdentifier())...
...;
    }

    public NodeTypes getNodeType(){
        return nodeType_;
    }

    public ArrayList<Node3D<Double>> getNeighbors(){
        ArrayList<Node3D<Double>> neighbors = new ArrayList<Node3D<Doub...
...le>>(outLinkList_.size());

        for(Link3D link:outLinkList_){
            neighbors.add(link.getToNode());
        }
        return neighbors;

    }

    // Neighbors for Node with check for NaN at index
    public ArrayList<Node3D<Double>> getNeighbors(int index){
        ArrayList<Node3D<Double>> neighbors = new ArrayList<Node3D<Doub...
...le>>(outLinkList_.size());

        for(Link3D link:outLinkList_){
            Node3D<Double> candidateNode = link.getToNode();
            if(!Double.isNaN(candidateNode.getDoubleValue(index))){
                neighbors.add(candidateNode);
            }
        }
```

```
        return neighbors;
    }


    public ArrayList<Link3D> getEdgesTo(Node3D<Double> node){
        ArrayList<Link3D> edgesTo = new ArrayList<Link3D>();

        for(Link3D link:outLinkList_){
            if(link.getToNode().equals(node)) edgesTo.add(link);
        }

        return edgesTo;
    }

    public ArrayList<Link3D> getOutLinkList(){
        return outLinkList_;
    }

    public synchronized Node3D<E> addLinkToOutLinkList(Link3D link, Str...
...ing linkId){
        for(Link3D outLink:outLinkList_){
            if(outLink.getIdentifier().equals(linkId)){
                return this;
            }
        }

        outLinkList_.add(link);
        return this;
    }

//    public double getCostAtNode(){
//        return costAtNode_;
//    }
//
//    public Node3D<E> setCostAtNode(double cost){
//        costAtNode_ = cost;
//        return this;
//    }
//
//    public Link3D getInLink(){
//        return inLink_;
//    }
//
//    public Node3D<E> setInLink(Link3D link){
//        inLink_ = link;
//        return this;
//    }
```

```
//
//    public SPAState getAppearedInQueueFlag(){
//        return appearedInQueue_;
//    }
//
//    public Node3D<E> setAppearedInQueueFlag(SPAState appeared){
//        appearedInQueue_ = appeared;
//        return this;
//    }

    public Node3D<E> setParentSector(ASector sector){
        parentSector_ = sector;
        return this;
    }

    public ASector getParentSector(){
        return parentSector_;
    }

    public Node3D<E> trimArrayLists(){
        outLinkList_.trimToSize();
        return this;
    }

    public boolean isAtFloor(){
        return nodeType_.equals(NodeTypes.FixFloor) ||
                nodeType_.equals(NodeTypes.NavaidFloor);

    }

    public boolean isAtCentroid(){
        return nodeType_.equals(NodeTypes.FixCentroid) ||
                nodeType_.equals(NodeTypes.NavaidCentroid);

    }

    public boolean isAtCeiling(){
        return nodeType_.equals(NodeTypes.FixCeiling) ||
                nodeType_.equals(NodeTypes.NavaidCeiling);
    }

    public boolean isAtBoundary(){
        return nodeType_.equals(NodeTypes.Boundary);
    }

    public boolean isAtStar(){
        return nodeType_.equals(NodeTypes.Star);
```

```
    }

    public boolean isAtAirport(){
        return nodeType_.equals(NodeTypes.Airport);
    }
}
```

NodeComparator.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author Jeff
 */
import java.util.*;
public class NodeComparator implements Comparator<Node3D<Double>> {
    private int labelIndex_;

    public NodeComparator(int labelIndex){
        labelIndex_ = labelIndex;
    }
    public int compare(Node3D<Double> node1, Node3D<Double> node2){
        double double1 = node1.getDoubleValue(labelIndex_);
        double double2 = node2.getDoubleValue(labelIndex_);

        if(double1 < double2) return -1;
        if(double1 == double2) return 0;
        return 1;
    }

}
```

PositionAtNode.java

```
/*
 * PositionAtNode.java
 *
 * Created on July 31, 2007, 11:51 AM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author jeff
 */
import java.util.*;
public class PositionAtNode {
    private Node3D<Double> node_;
    private double timeMinutes_;
    private double distanceNm_;
    private double altitudeFl_;

    /** Creates a new instance of PositionAtNode */
    public PositionAtNode() {
    }

    public PositionAtNode(Node3D<Double> node, double timeMinutes,
            double distanceNm, double altitudeFl){
        node_ = node;
        timeMinutes_ = timeMinutes;
        distanceNm_ = distanceNm;
        altitudeFl_ = altitudeFl;
    }

    public Node3D<Double> getNode(){
        return node_;
    }

    public double getTimeMinutes(){
        return timeMinutes_;
    }

    public double getDistanceNm(){
        return distanceNm_;
    }

    public double getAltitudeFl(){
        return altitudeFl_;
```

```
    }

}
```

Scenario.java

```
/*
 * Scenario.java
 *
 * Created on July 25, 2007, 1:44 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author jeff
 */
import java.util.*;
import java.util.concurrent.*;
public class Scenario implements Comparable {
    private Network network_;
    private Airspace airspace_;
    private String name_;

    /* Index is defined in the sector object. First index is the sector...
....
     * Second index is the time period.
     */
    private double[][] sectorCapacity_;
    private double[][] sectorFlowByMin_;

    /** Flight path data and demand by node. */
    private ConcurrentHashMap<String, ArrayList<PositionAtNode>> flight...
...Path_;
    private ConcurrentHashMap<String, HashMap<String, Integer>> nodeInd...
...exForFlight_;
//    private HashMap<String, ArrayList<String>> flightsUsingNode_;

    /** Flight path demand by network link */
    private ConcurrentHashMap<String, ArrayList<Link3D>> flightLinks_;
    private ConcurrentHashMap<String, HashMap<String, Integer>> linkInd...
...exForFlight_;
//    private HashMap<String, ArrayList<String>> flightsUsingLink_;

    /** Demand by time period for flight */
    private ConcurrentHashMap<String, HashMap<Integer, Double>> flightD...
...emandTimePeriod_;

    /** flightPath_ index for clustering  */
    private ConcurrentHashMap<String, Integer> flightStartClusterIndex_...
```

```
...;
    private ConcurrentHashMap<String, Integer> flightEndClusterIndex_;

    /** Random weather data */
    private ArrayList<GPoint2D<Double>> randomConvectionLocations_;
    private ArrayList<Double> randomConvectionAreas10nm_;
    private ArrayList<ArrayList<Double>> randomConvectionLatitudes10nm_...
...;
    private ArrayList<ArrayList<Double>> randomConvectionLongitudes10nm...
..._;
    private ArrayList<Double> randomConvectionAreas20nm_;
    private ArrayList<ArrayList<Double>> randomConvectionLatitudes20nm_...
...;
    private ArrayList<ArrayList<Double>> randomConvectionLongitudes20nm...
..._;
    private ArrayList<Double> randomConvectionTops_;
    private ArrayList<Integer> randomStartTimeMin_;
    private ArrayList<Integer> randomEndTimeMin_;

    /** Capacity constrained sectors (i.e. in flow constrained area)*/
    private ArrayList<ASector> constrainedSectors_;

    /** Clustering results */
    private ConcurrentHashMap<String, Integer> clusterIndexForFlight_;
    private ConcurrentHashMap<Integer, ArrayList<String>> flightsInClus...
...ter_;
    private ConcurrentHashMap<Integer, Node3D<Double>> clusterStartMedo...
...id_;
    private ConcurrentHashMap<Integer, Node3D<Double>> clusterEndMedoid...
..._;

    /** k-shortest paths result */
    private ConcurrentHashMap<Integer, ArrayList<ArrayList<Link3D>>> kS...
...hortestPathsByCluster_;

    /** Incremental assignment result */
    private ConcurrentHashMap<Integer, ArrayList<Link3D>> incrementalPa...
...thsByCluster_;

    /** Check that assignment length is not too long */
    private ConcurrentHashMap<Integer, Boolean> useFlightPlan_;

    private static int numberSectors_;

    /** Cache of factorial */
    private ArrayList<Double> factorials_;
```

```
    /** Creates a new instance of Scenario */
    public Scenario(String name, Network network, Airspace airspace) {
        network_ = network;
        airspace_ = airspace;
        name_ = name;

        // Initialize path for flights (node-based)
        flightPath_ = new ConcurrentHashMap<String, ArrayList<PositionA...
...tNode>>();
        nodeIndexForFlight_ = new ConcurrentHashMap<String, HashMap<Str...
...ing, Integer>>();
//        flightsUsingNode_ = new HashMap<String, ArrayList<String>>(
//                Settings.hashMapDefaultInitialCapacity,
//                Settings.hashMapLoadFactor);

        // Initialize link-based data
        flightLinks_ = new ConcurrentHashMap<String, ArrayList<Link3D>>...
...();
        linkIndexForFlight_ = new ConcurrentHashMap<String, HashMap<Str...
...ing, Integer>>();
//        flightsUsingLink_ = new HashMap<String, ArrayList<String>>(
//                Settings.hashMapDefaultInitialCapacity,
//                Settings.hashMapLoadFactor);

        // Clustering data
        flightStartClusterIndex_ = new ConcurrentHashMap<String, Intege...
...r>(
                Settings.hashMapDefaultInitialCapacity);
        flightEndClusterIndex_ = new ConcurrentHashMap<String, Integer>...
...(
                Settings.hashMapDefaultInitialCapacity);

        // Initialize weather data
        randomConvectionLocations_ = new ArrayList<GPoint2D<Double>>();...
...        randomConvectionAreas10nm_ = new ArrayList<Double>();
        randomConvectionLatitudes10nm_ = new ArrayList<ArrayList<Double...
...>>();
        randomConvectionLongitudes10nm_ = new ArrayList<ArrayList<Doubl...
...e>>();
        randomConvectionAreas20nm_ = new ArrayList<Double>();
        randomConvectionLatitudes20nm_ = new ArrayList<ArrayList<Double...
...>>();
        randomConvectionLongitudes20nm_ = new ArrayList<ArrayList<Doubl...
...e>>();
        randomConvectionTops_ = new ArrayList<Double>();
        randomStartTimeMin_ = new ArrayList<Integer>();
        randomEndTimeMin_ = new ArrayList<Integer>();
```

```
        // Capacity constrained sectors
        constrainedSectors_ = new ArrayList<ASector>();

        // Demand indexed by flight
        flightDemandTimePeriod_ = new ConcurrentHashMap<String,HashMap<...
...Integer,Double>>();

        // Initialize cluster data
        clusterIndexForFlight_ = new ConcurrentHashMap<String, Integer>...
...(
                Settings.hashMapDefaultInitialCapacity);
        flightsInCluster_ = new ConcurrentHashMap<Integer, ArrayList<St...
...ring>>(
                Settings.hashMapDefaultInitialCapacity);
        clusterStartMedoid_ = new ConcurrentHashMap<Integer, Node3D<Dou...
...ble>>(
                Settings.hashMapDefaultInitialCapacity);
        clusterEndMedoid_ = new ConcurrentHashMap<Integer, Node3D<Doubl...
...e>>(
                Settings.hashMapDefaultInitialCapacity);

        kShortestPathsByCluster_ = new ConcurrentHashMap<Integer, Array...
...List<ArrayList<Link3D>>>(
                Settings.hashMapDefaultInitialCapacity);

        incrementalPathsByCluster_ = new ConcurrentHashMap<Integer, Arr...
...ayList<Link3D>>(
                Settings.hashMapDefaultInitialCapacity);

        useFlightPlan_ = new ConcurrentHashMap<Integer, Boolean>(this.n...
...oClusters());

        factorials_ = new ArrayList<Double>(100);

    }

    public Scenario clearClusteringResults(){
        clusterIndexForFlight_ = new ConcurrentHashMap<String, Integer>...
...(
                Settings.hashMapDefaultInitialCapacity);
        flightsInCluster_ = new ConcurrentHashMap<Integer, ArrayList<St...
...ring>>(
                Settings.hashMapDefaultInitialCapacity);
        clusterStartMedoid_ = new ConcurrentHashMap<Integer, Node3D<Dou...
...ble>>(
                Settings.hashMapDefaultInitialCapacity);
```

```
        clusterEndMedoid_ = new ConcurrentHashMap<Integer, Node3D<Doubl...
...e>>(
                Settings.hashMapDefaultInitialCapacity);


        return this;
    }

    public Scenario addRandomConvection(GPoint2D<Double> location, doub...
...le area10,
            double[] latitudes10, double[] longitudes10, double area20,...
...            double[] latitudes20, double[] longitudes20, double top,
            int startTimeMin, int endTimeMin){
        randomConvectionLocations_.add(location);
        randomConvectionAreas10nm_.add(area10);
        randomConvectionAreas20nm_.add(area20);
        randomConvectionTops_.add(top);
        randomStartTimeMin_.add(startTimeMin);
        randomEndTimeMin_.add(endTimeMin);

        ArrayList<Double> latArray10 = new ArrayList<Double>(latitudes1...
...0.length);
        ArrayList<Double> lonArray10 = new ArrayList<Double>(longitudes...
...10.length);

        for(int i=0; i<latitudes10.length; i++){
            latArray10.add(latitudes10[i]);
            lonArray10.add(longitudes10[i]);
        }

        randomConvectionLatitudes10nm_.add(latArray10);
        randomConvectionLongitudes10nm_.add(lonArray10);


        ArrayList<Double> latArray20 = new ArrayList<Double>(latitudes2...
...0.length);
        ArrayList<Double> lonArray20 = new ArrayList<Double>(longitudes...
...20.length);

        for(int i=0; i<latitudes20.length; i++){
            latArray20.add(latitudes20[i]);
            lonArray20.add(longitudes20[i]);
        }

        randomConvectionLatitudes20nm_.add(latArray20);
        randomConvectionLongitudes20nm_.add(lonArray20);
```

```
        return this;
    }

    /** Sector capacity is in 15 minute intervals */
    public double getSectorCapacity15Min(int sectorIndex, int timePerio...
...d){
        return sectorCapacity_[sectorIndex][timePeriod];
    }

    /** Sector capacity is in 15 minute intervals */
    public synchronized Scenario setSectorCapacity15Min(int sectorIndex...
..., int timePeriod, double sectorCapacity){
        sectorCapacity_[sectorIndex][timePeriod] = sectorCapacity;
        return this;
    }

    public int noRandomConvectionLocations(){
        return randomConvectionLocations_.size();
    }

    public GPoint2D<Double> getRandomConvectionLocation(int idx){
        return randomConvectionLocations_.get(idx);
    }

    public double getRandomConvectionArea10nm(int idx){
        return randomConvectionAreas10nm_.get(idx);
    }

    public double getRandomConvectionArea20nm(int idx){
        return randomConvectionAreas20nm_.get(idx);
    }

    public double[] getRandomConvectionLatitudes10nm(int idx){
        ArrayList<Double> latArray = randomConvectionLatitudes10nm_.get...
...(idx);
        double[] latArrayType = new double[latArray.size()];
        for(int i=0; i<latArray.size(); i++){
            latArrayType[i] = latArray.get(i);
        }
        return latArrayType;
    }

    public double[] getRandomConvectionLongitudes10nm(int idx){
        ArrayList<Double> lonArray = randomConvectionLongitudes10nm_.ge...
...t(idx);
        double[] lonArrayType = new double[lonArray.size()];
        for(int i=0; i<lonArray.size(); i++){
```

```
            lonArrayType[i] = lonArray.get(i);
        }
        return lonArrayType;
    }

    public double[] getRandomConvectionLatitudes20nm(int idx){
        ArrayList<Double> latArray = randomConvectionLatitudes20nm_.get...
...(idx);
        double[] latArrayType = new double[latArray.size()];
        for(int i=0; i<latArray.size(); i++){
            latArrayType[i] = latArray.get(i);
        }
        return latArrayType;
    }

    public double[] getRandomConvectionLongitudes20nm(int idx){
        ArrayList<Double> lonArray = randomConvectionLongitudes20nm_.ge...
...t(idx);
        double[] lonArrayType = new double[lonArray.size()];
        for(int i=0; i<lonArray.size(); i++){
            lonArrayType[i] = lonArray.get(i);
        }
        return lonArrayType;
    }

    public double getRandomConvectionTop(int idx){
        return randomConvectionTops_.get(idx);
    }

    public int getRandomStartTime(int idx){
        return randomStartTimeMin_.get(idx);
    }

    public int getRandomEndTime(int idx){
        return randomEndTimeMin_.get(idx);
    }

    public Scenario addFlightLink(String flightId, String linkId, Link3...
...D link){
        // Processing to update 'flightLinks_'
        ArrayList<Link3D> links = flightLinks_.containsKey(flightId) ?
            flightLinks_.get(flightId) : new ArrayList<Link3D>();
        links.add(link);
        flightLinks_.put(flightId, links);

        // Processing to update 'linkIndexForFlight_'
        HashMap<String, Integer> linkIndex = linkIndexForFlight_.contai...
```

```
...nsKey(flightId) ?
            linkIndexForFlight_.get(flightId) : new HashMap<String, Int...
...eger>(
                Settings.hashMapDefaultInitialCapacity,
                Settings.hashMapLoadFactor);
        linkIndex.put(linkId, links.size() - 1);
        linkIndexForFlight_.put(flightId, linkIndex);

//        // Processing to update 'flightsUsingLink_'
//        ArrayList<String> flights = flightsUsingLink_.containsKey(lin...
...kId) ?
//            flightsUsingLink_.get(linkId) : new ArrayList<String>();
//        if(!flights.contains(flightId)) flights.add(flightId);
//        flightsUsingLink_.put(linkId, flights);


        return this;
    }

    public ArrayList<Double> getSectorTraversalTimes(String flightId){
        if(flightPath_.containsKey(flightId)){
            HashMap<String, Double> sectorTimes = new HashMap<String, D...
...ouble>();

            ArrayList<PositionAtNode> flightPositions = flightPath_.get...
...(flightId);

            for(int i=0; i<flightPositions.size(); i++){
                if(i == 0) continue;
                PositionAtNode pan1 = flightPositions.get(i-1);
                PositionAtNode pan2 = flightPositions.get(i);

                Node3D<Double> node1 = pan1.getNode();
                Node3D<Double> node2 = pan2.getNode();

                ASector sec1 = node1.getParentSector();
                ASector sec2 = node2.getParentSector();

                // Do not add null sectors
                if(sec1.equals(ASector.NullSector) ||
                        sec2.equals(ASector.NullSector)) continue;

                if(sec1.equals(sec2)){
                    String sectorName = sec1.getName();

                    double timeDiff = Math.abs(pan2.getTimeMinutes() -
                            pan1.getTimeMinutes());
```

```
                    if(sectorTimes.containsKey(sectorName)){
                        double timeDiff1 = sectorTimes.get(sectorName);...
...                        timeDiff += timeDiff1;
                    }
                    sectorTimes.put(sectorName, timeDiff);
                }
            }

            ArrayList<Double> returnTimes = new ArrayList<Double>(secto...
...rTimes.size());
            for(double val:sectorTimes.values()){
                returnTimes.add(val);
            }

            return returnTimes;
        }else{
            return new ArrayList<Double>(1);
        }
    }

    public Link3D getFlightLink(String flightId, int index){
        ArrayList<Link3D> links = flightLinks_.get(flightId);
        return links.get(index);
    }

    public int noFlightLinks(String flightId){
        if(!flightLinks_.containsKey(flightId)) return 0;
        ArrayList<Link3D> links = flightLinks_.get(flightId);
        return links.size();
    }

    public Scenario addFlightPosition(String flightId, String nodeId,
            Node3D<Double> node,
            double timeMinutes, double distanceNm, double altitudeFl,
            TraversalTimeDistribution ttd){
        // Begin code for 'flightPath_'
        // Create list of locations if it does not already exist
        ArrayList<PositionAtNode> flightPositions =
                flightPath_.containsKey(flightId) ?
                flightPath_.get(flightId) :
                new ArrayList<PositionAtNode>();

        // Object to store position at the Node
        PositionAtNode pan = new PositionAtNode(node, timeMinutes, dist...
...anceNm,
                altitudeFl);
```

```
        // Store the results
        flightPositions.add(pan);
        flightPath_.put(flightId, flightPositions);


        // Begin code for 'nodeIndexForFlight_'
        HashMap<String, Integer> nodeIndices =
                nodeIndexForFlight_.containsKey(flightId) ?
                nodeIndexForFlight_.get(flightId) :
                new HashMap<String, Integer>();
        nodeIndices.put(nodeId, flightPositions.size() - 1);
        nodeIndexForFlight_.put(flightId, nodeIndices);

//        // Begin code for 'flightsUsingNode_'
//        ArrayList<String> flightIds =
//                flightsUsingNode_.containsKey(nodeId) ?
//                flightsUsingNode_.get(nodeId) :
//                new ArrayList<String>();
//        if(!flightIds.contains(flightId)) flightIds.add(flightId);
//        flightsUsingNode_.put(nodeId, flightIds);

        // Probabilistic assignment of flights
        //
        // Do not add demand at the first position
        if(flightPositions.size() == 1) return this;

        // Get previous positions
        PositionAtNode previousPan = flightPositions.get(flightPosition...
...s.size() - 2); // - 1);

        // Only add demand if some time has elapsed
        double elapsedTimeMinutes = timeMinutes - previousPan.getTimeMi...
...nutes();
        long previousTime = Math.round(previousPan.getTimeMinutes());
        long currentTime = Math.round(timeMinutes);
        if(previousTime == currentTime) return this;

        ASector toNodeSector = node.getParentSector();
        KernelDensity errorDistribution  = ttd.run(toNodeSector, elapse...
...dTimeMinutes);

        ArrayList<Double> errorX = errorDistribution.getValues();
        ArrayList<Double> density = errorDistribution.getDensities();

        for(long i2 = previousTime; i2<=currentTime; i2++){
            for(int j=0; j<errorX.size(); j++){
```

```
                double timeToAdd = i2 + errorX.get(j);
                double valToAdd = density.get(j);
                incrementSectorFlow(flightId, toNodeSector, timeToAdd, ...
...valToAdd);
            }
        }

        return this;
    }

    public int getNumberFlightPositions(String flightId){
        if(!flightPath_.containsKey(flightId)){
            return 0;
        }else{
            return flightPath_.get(flightId).size();
        }
    }

    public PositionAtNode getFlightPosition(String flightId, int index)...
...{
        ArrayList<PositionAtNode> flightPositions = flightPath_.get(fli...
...ghtId);
        return flightPositions.get(index);
    }

    public PositionAtNode getFlightPosition(String flightId, String nod...
...eId){
        HashMap<String, Integer> nodeIndices = nodeIndexForFlight_.get(...
...flightId);
        int nodeIndex = nodeIndices.get(nodeId);
        return getFlightPosition(flightId, nodeIndex);
    }

    /** Name of the scenario (e.g. flight plan) */
    public String getName(){
        return name_;
    }

    /** Convert time in minutes to a time period (integer index) */
    private int convertTimeToTimePeriod(double timeD){
        double timeDFloor = Math.floor(timeD/15.0D);
        long timeL = Math.round(timeDFloor);

        return (int)timeL;
    }

    /** Convert time in minutes to a time period index by minute */
```

```
    private int convertTimeToTimePeriodByMin(double timeD){
        long timeL = Math.round(timeD);

        return (int)timeL;
    }

    public Scenario incrementSectorFlow(String flightId, ASector sector...
...,
            double[] times, double[] valsToAdd){
        for(int i=0; i<times.length; i++){
            double time = times[i];
            double valToAdd = valsToAdd[i];

            incrementSectorFlow(flightId, sector, time, valToAdd);
        }

        return this;
    }

    public Scenario incrementSectorFlow(String flightId, ASector sector...
...,
            double time, double valToAdd){
        int timeIndex = this.convertTimeToTimePeriodByMin(time);

        if(timeIndex >= Settings.numAnalysisTimePeriodsOneMin - 1
                || timeIndex < 0) return this;

        int sectorIndex = sector.getArrayIndex();

        if(sectorIndex < 0) return this;

        sectorFlowByMin_[sectorIndex][timeIndex] += valToAdd;
        addFlightDemand(flightId, timeIndex, valToAdd, sectorIndex);

        return this;
    }



    private Scenario addFlightDemand(String flightId, int timeIndex,
            double demand, int sectorIndex){
        HashMap<Integer, Double> timeDemand = null;

        if(flightDemandTimePeriod_.containsKey(flightId)){
            timeDemand = flightDemandTimePeriod_.get(flightId);
        }else{
            timeDemand = new HashMap<Integer, Double>(
```

```
                    Settings.hashMapDefaultInitialCapacity,
                    Settings.hashMapLoadFactor);
        }

        int index = timeIndex + sectorIndex * 10000;

        if(timeDemand.containsKey(index)){
            double currentDemand = timeDemand.get(index);
            currentDemand += demand;
            timeDemand.put(index, currentDemand);
        }else{
            timeDemand.put(index, demand);
        }

        return this;
    }

    public Scenario incrementSectorFlow(String flightId, ASector sector...
...,
            double startTime, double endTime, double valToAdd){
        // Get the current time interval
        int startTimeIndex = this.convertTimeToTimePeriodByMin(startTim...
...e);
        int endTimeIndex = this.convertTimeToTimePeriodByMin(endTime);
        // Check to see if start and/or end times go beyond the end of ...
...the scenario
        if(startTimeIndex >= Settings.numAnalysisTimePeriodsOneMin - 1 ...
...||
                endTimeIndex >= Settings.numAnalysisTimePeriodsOneMin -...
... 1 ||
                startTimeIndex < 0 || endTimeIndex < 0)
            return this;

        if(startTimeIndex == endTimeIndex) return this;

        // Get the sector index
        int sectorIndex = sector.getArrayIndex();

        if(sectorIndex < 0) return this;

        for(int timeIndex = startTimeIndex; timeIndex <= endTimeIndex; ...
...timeIndex++){
            // Update the flow
            sectorFlowByMin_[sectorIndex][timeIndex] += valToAdd;
            addFlightDemand(flightId, timeIndex, valToAdd, sectorIndex)...
...;
        }
```

```
        return this;
    }

    public double getSectorDemand(int sectorIdx, int timeMinutes){
        return sectorFlowByMin_[sectorIdx][timeMinutes];
    }

    public double getSectorCapacity15minInterval(int sectorIdx, int tim...
...eInterval){
        return sectorCapacity_[sectorIdx][timeInterval];
    }

    public double[][] getSectorDemandArray(){
        return sectorFlowByMin_;
    }

    public Scenario setSectorDemandArray(double[][] sectorFlowByMin){
        for(int i=0; i<numberSectors_; i++){
            for(int j=0; j<Settings.numAnalysisTimePeriodsOneMin; j++){...
...                 sectorFlowByMin_[i][j] = sectorFlowByMin[i][j];
            }
        }
        //sectorFlowByMin_ = sectorFlowByMin;
        return this;
    }

    public double[][] getSectorCapacityArray(){
        return sectorCapacity_;
    }

    public Scenario setSectorCapacityArray(double[][] sectorCapacity){
        sectorCapacity_ = sectorCapacity;
        return this;
    }

    /** Get total demand (aircraft-minutes) */
    public double totalSectorDemand(){
        double tot = 0.0D;

        for(int i = 0; i < airspace_.getNoSectors(); i++){
            for(int j=0; j < Settings.numAnalysisTimePeriodsOneMin; j++...
...){
                tot += sectorFlowByMin_[i][j];
            }
        }
```

```
        return tot;
    }

    /** Initialize capacity to default capacity and flow to zero. */
    public Scenario initializeSectorCapacityAndFlow(){
        numberSectors_ = 0;

        // Get the number of sectors
        for(String centerS:FAPio.writeCenters){
            ACenter center = airspace_.getCenter(centerS);
            Iterator<ASector> e3 = center.getSectorValueIterator();
            while(e3.hasNext()){
                ASector sector = e3.next();
                int sectorIndex = sector.getArrayIndex();
                if(sectorIndex < 0) continue;
                numberSectors_++;
            }
        }

        // Initialize sector arrays
        sectorCapacity_ = new double[numberSectors_]
                [Settings.numAnalysisTimePeriods];
        sectorFlowByMin_ = new double[numberSectors_]
                [Settings.numAnalysisTimePeriodsOneMin];

        // Iterate through the sectors
        for(String centerS:FAPio.writeCenters){
            ACenter center = airspace_.getCenter(centerS);
            Iterator<ASector> e3 = center.getSectorValueIterator();
            while(e3.hasNext()){
                ASector sector = e3.next();
                int sectorIndex = sector.getArrayIndex();
                if(sectorIndex < 0) continue;
                int sectorMap = sector.getMapValue();

                // Capacity is in 15 minute intervals;
                for(int i=0; i<Settings.numAnalysisTimePeriods; i++){
                    sectorCapacity_[sectorIndex][i] = sectorMap;
                }

                // Flow (demand) is in 1 minute intervals
                for(int i=0; i<Settings.numAnalysisTimePeriodsOneMin; i...
...++){
                    sectorFlowByMin_[sectorIndex][i] = 0.0D;
                }
            }
        }
```

```
        return this;
    }

    /** Get sectors that are over capacity a significant portion of tim...
...e */
    public ArrayList<ASector> calculateFlowConstrainedSectors(int start...
...TimeMin, int endTimeMin){
        constrainedSectors_ = new ArrayList<ASector>();

        Airspace a = APData.airspace;
        Iterator<ACenter> e = a.getCenterValueIterator();
        while(e.hasNext()){
            ACenter center = e.next();

            Iterator<ASector> e2 = center.getSectorValueIterator();
            while(e2.hasNext()){
                ASector sector = e2.next();

                int idx = sector.getArrayIndex();

                if(idx < 0) continue;

                double numTimePeriodsExceedingCapacityThreshold = 0;
                double totTimePeriods = 0;

                for(int i=startTimeMin; i<=endTimeMin; i++){
                    totTimePeriods++;

                    double demand = sectorFlowByMin_[idx][i];

                    double i2 = (double)i;
                    double timeP = Math.round(i2/15 - i2%15);
                    int timeI = (int)timeP;

                    double capacity = sectorCapacity_[idx][timeI];

                    double pec = poissonProbabilityExceedingCapacity(de...
...mand, capacity);

                    if(pec >= Settings.capacityProbabilityThreshold){
                        numTimePeriodsExceedingCapacityThreshold++;
                    }
                }

                double timeCongested = numTimePeriodsExceedingCapacityT...
...hreshold/totTimePeriods;
```

```
                if(timeCongested > Settings.proportionTimeCongested){
                    constrainedSectors_.add(sector);
                }
            }
        }

        return constrainedSectors_;
    }

    /** Clustering of flights */
    public Scenario clusterFlights(){
        // Distance from flow constrained area (FCA) to begin clusterin...
...g (nm)
        final int CLUSTERDISTANCETOFCA = 300;

        Network n = APData.network;
        Airspace a = APData.airspace;

        // Calculate flights passing through the flow constrained area
        n.calculateFlightsInFlowConstrainedArea();
        ArrayList<Flight> constrainedFlightsLT18kft =
                n.getConstrainedFlightsLessThan18kft();
        ArrayList<Flight> constrainedFlightsGT18kft =
                n.getConstrainedFlightsGreater18kft();

        // Sort the lists for consistency
        Collections.sort(constrainedFlightsLT18kft);
        Collections.sort(constrainedFlightsGT18kft);

        // Get the latitude/longitude of the flow constrained area
        double[] fcaLatitude = n.getConstrainedSectorsLat();
        double[] fcaLongitude = n.getConstrainedSectorsLon();

        for(int c=1; c<=2; c++){
            ArrayList<Flight> constrainedFlights = null;
            int numClusters = 0;
            if(c == 1){
                // First do flights < 18 kft
                constrainedFlights = constrainedFlightsLT18kft;
                //numClusters = Settings.numberFlightClustersLT18kft;
            }else{
                // then flights >= 18 kft
                constrainedFlights = constrainedFlightsGT18kft;
                //numClusters = Settings.numberFlightClustersGT18kft;
            }
```

```
            // Latitude/Longitudes used for clustering
            double[] flightStartLat = new double[constrainedFlights.siz...
...e()];
            double[] flightStartLon = new double[constrainedFlights.siz...
...e()];
            double[] flightEndLat = new double[constrainedFlights.size(...
...)];
            double[] flightEndLon = new double[constrainedFlights.size(...
...)];

            for(int i=0; i<constrainedFlights.size(); i++){
                Flight flight = constrainedFlights.get(i);
                String flightId = flight.getCallSign();

                // Latitude/Longitude of flight plan
                double[] flLats = this.getConstrainedFlightLat(flightId...
...);
                double[] flLons = this.getConstrainedFlightLon(flightId...
...);
                // Calculate distance from each waypoint to the flow co...
...nstrained area
                double[] flDists = new double[flLats.length];
                for(int j=0; j<flLats.length; j++){
                    double minDist = Double.MAX_VALUE;
                    for(int k=0; k<fcaLatitude.length; k++){
                        if(Double.isNaN(fcaLatitude[k]) ||
                                Double.isNaN(fcaLongitude[k])) continue...
...;
                        Mapping m = new Mapping(flLats[j],flLons[j],
                                fcaLatitude[k],fcaLongitude[k]);
                        minDist = Math.min(minDist,m.getDistance());
                    }
                    flDists[j] = minDist;
                }

                // Find the closest point greater than clusterDistanceT...
...oFCA
                // to flow constrained area at the beginning and end of...
... the path.
                int startIndex = 0;
                for(int j=0; j<flDists.length; j++){
                    if(flDists[0] <= CLUSTERDISTANCETOFCA) break;
                    if(flDists[j] > CLUSTERDISTANCETOFCA) startIndex = ...
...j;
                }

                int lengthflDists = flDists.length;
```

```
                int endIndex = lengthflDists - 1;
                for(int j=0; j<flDists.length; j++){
                    int j2 = lengthflDists - j - 1;
                    if(flDists[j2] <= CLUSTERDISTANCETOFCA) break;
                    else endIndex = j2;
                }

                // Store the indices
                this.setFlightStartClusterIndex(flightId, startIndex);
                this.setFlightEndClusterIndex(flightId, endIndex);

                flightStartLat[i] = flLats[startIndex];
                flightStartLon[i] = flLons[startIndex];
                flightEndLat[i] = flLats[endIndex];
                flightEndLon[i] = flLons[endIndex];
            }

            // Cluster flights
            double[][] flCoordCluster = new double[flightStartLat.lengt...
...h][4];
            for(int i=0; i<flightStartLat.length; i++){
                flCoordCluster[i][0] = flightStartLat[i];
                flCoordCluster[i][1] = flightStartLon[i];
                flCoordCluster[i][2] = flightEndLat[i];
                flCoordCluster[i][3] = flightEndLon[i];
            }



//            double[][] T = MatlabDriver.completeCluster(flCoordCluste...
...r);
//
//            // Get the number of clusters
//            numClusters = 0;
//            int[] T2 = new int[T.length];
//            for (int i = 0; i < T.length; i++) {
//                Double tmpNumClust = T[i][0];
//                T2[i] = tmpNumClust.intValue();
//                numClusters = Math.max(numClusters, T2[i]);
//            }

            int[] T2 = clusterData(flCoordCluster);

            // Get the number of clusters
            numClusters = 0;
            for(int i=0; i<T2.length; i++) numClusters = Math.max(numCl...
...usters, T2[i]);
```

```
            if(c==1){
                Settings.numberFlightClustersLT18kft = numClusters;
                System.out.println("Clusters for flights cruising below...
... FL 180: "
                        + Integer.toString(numClusters));
            }else{
                Settings.numberFlightClustersGT18kft = numClusters;
                System.out.println("Clusters for flights cruising above...
... FL 180: "
                        + Integer.toString(numClusters));
            }

            for (int i = 0; i < T2.length; i++) {
                // Create a unique index
                if (c == 2) {
                    T2[i] += Settings.numberFlightClustersLT18kft;
                }
                // Add the index for each flight
                Flight flight = constrainedFlights.get(i);
                String flightId = flight.getCallSign();
                this.addClusterIndexToFlight(flightId, T2[i]);
            }

            // Get the medoids
            for(int i=0; i<numClusters; i++){
                int[] T3 = new int[T2.length];
                for(int k=0; k<T2.length; k++){
                    T3[k] = T2[k];
                }

                double[] startLatCpy = new double[flightStartLat.length...
...];
                double[] startLonCpy = new double[flightStartLon.length...
...];
                double[] endLatCpy = new double[flightEndLat.length];
                double[] endLonCpy = new double[flightEndLon.length];
                for(int k=0; k<flightStartLat.length; k++){
                    startLatCpy[k] = flightStartLat[k];
                    startLonCpy[k] = flightStartLon[k];
                    endLatCpy[k] = flightEndLat[k];
                    endLonCpy[k] = flightEndLon[k];
                }

                int idx = c==2 ? i + Settings.numberFlightClustersLT18k...
...ft + 1 : i + 1;
```

```
                for(int j=0; j<T3.length; j++){
                    if(T3[j] == idx){
                        continue;
                    }

                    T3[j] = 0;
                    startLatCpy[j] = 0D;
                    startLonCpy[j] = 0D;
                    endLatCpy[j] = 0D;
                    endLonCpy[j] = 0D;
                }

                // Get medoid for start
                double minStartDist = Double.MAX_VALUE;
                int minStartIdx = -1;
                for(int j=0; j<T3.length; j++){
                    if(T3[j] != idx) continue;

                    double tmpDist = 0D;
                    for(int k=0; k<startLatCpy.length; k++){
                        if(T3[k] != idx) continue;
                        Mapping m = new Mapping(startLatCpy[j], startLo...
...nCpy[j],
                                startLatCpy[k], startLonCpy[k]);
                        tmpDist += m.getDistance();
                    }

                    if(tmpDist < minStartDist){
                        minStartDist = tmpDist;
                        minStartIdx = j;
                    }
                }
//                if(minStartIdx < 0){
//                    ArrayList<Double> T3_1 = new ArrayList<Double>();...
...//                    ArrayList<Double> T3_2 = new ArrayList<Double>();...
...//
//                    for(int j=0; j<T3.length; j++){
//                        if(T3[j] != idx) continue;
//
//                        T3_1.add(startLatCpy[j]);
//                        T3_2.add(startLonCpy[j]);
//                    }
//
//                    System.out.println(minStartIdx);
//                }

                Flight flight = constrainedFlights.get(minStartIdx);
```

```
                String flightId = flight.getCallSign();
                this.addClusterStartMedoid(idx, flightId);

                // Get medoid for end
                double minEndDist = Double.MAX_VALUE;
                int minEndIdx = -1;
                for(int j=0; j<T3.length; j++){
                    if(T3[j] != idx) continue;

                    double tmpDist = 0D;
                    for(int k=0; k<endLatCpy.length; k++){

                        Mapping m = new Mapping(endLatCpy[j], endLonCpy...
...[j],
                                endLatCpy[k], endLonCpy[k]);
                        tmpDist += m.getDistance();
                    }

                    if(tmpDist < minEndDist){
                        minEndDist = tmpDist;
                        minEndIdx = j;
                    }
                }
                flight = constrainedFlights.get(minEndIdx);
                flightId = flight.getCallSign();
                this.addClusterEndMedoid(idx, flightId);
            }

        }

        return this;
    }

    private int[] clusterData(double[][] flCoords){
        final int MAXCLUSTERINPUT = 2000;

        if(flCoords.length < MAXCLUSTERINPUT){
            double[][] T = MatlabDriver.completeCluster(flCoords);
            return clusterData1D(T);
        }else{
            ArrayList<double[][]> T_split = new ArrayList<double[][]>()...
...;
            int cnt = 0;
            ArrayList<Double> val0 = new ArrayList<Double>(MAXCLUSTERIN...
...PUT);
            ArrayList<Double> val1 = new ArrayList<Double>(MAXCLUSTERIN...
...PUT);
```

```
            ArrayList<Double> val2 = new ArrayList<Double>(MAXCLUSTERIN...
...PUT);
            ArrayList<Double> val3 = new ArrayList<Double>(MAXCLUSTERIN...
...PUT);
            for(int i=0; i<flCoords.length; i++){
                val0.add(flCoords[i][0]);
                val1.add(flCoords[i][1]);
                val2.add(flCoords[i][2]);
                val3.add(flCoords[i][3]);

                if(cnt++ >= MAXCLUSTERINPUT || i == flCoords.length - 1...
...){
                    double[][] val4 = new double[val0.size()][4];
                    for(int j=0; j<val0.size(); j++){
                        val4[j][0] = val0.get(j);
                        val4[j][1] = val1.get(j);
                        val4[j][2] = val2.get(j);
                        val4[j][3] = val3.get(j);
                    }
                    T_split.add(val4);

                    cnt = 0;
                    val0 = new ArrayList<Double>(MAXCLUSTERINPUT);
                    val1 = new ArrayList<Double>(MAXCLUSTERINPUT);
                    val2 = new ArrayList<Double>(MAXCLUSTERINPUT);
                    val3 = new ArrayList<Double>(MAXCLUSTERINPUT);
                }
            }

            ArrayList<int[]> T_split2 = new ArrayList<int[]>(T_split.si...
...ze());
            for(int i=0; i < T_split.size(); i++){
                double[][] flCoords4 = T_split.get(i);

                int[] T_res = clusterData1D(
                        MatlabDriver.completeCluster(flCoords4));

                T_split2.add(T_res);
            }
            T_split = null;

            int lenOut = 0;
            for(int[] arr:T_split2){
                lenOut += arr.length;
            }

            int[] T_return = new int[lenOut];
```

```
            int currAddIdx = 0;
            int idx = 0;
            for(int i=0; i<T_split2.size(); i++){
                int maxIdx = 0;

                int[] T_res = T_split2.get(i);
                for(int j=0; j<T_res.length; j++){
                    T_return[idx] = T_res[j] + currAddIdx;
                    maxIdx = Math.max(maxIdx, T_return[idx]);
                    idx++;
                }
                currAddIdx = maxIdx;
            }

            return T_return;
        }

    }

    private int[] clusterData1D(double[][] T){
        int[] T2 = new int[T.length];
        for (int i = 0; i < T.length; i++) {
            Double tmpNumClust = T[i][0];
            T2[i] = tmpNumClust.intValue();
        }

        return T2;
    }

    // Given a demand calculate the probability of exceeding capacity a...
...ssuming a
    // Poisson distribution.
    public double poissonProbabilityExceedingCapacity(double demand, do...
...uble capacity){
        if(demand <= 0.0D) return 0.0D;
        if(capacity == 0.0D) return 1.0D;
        if(demand >= capacity) return 1.0D;

        double k = Math.floor(capacity);

        double probLTCapacity = 0;

        for(int i=0; i<=k; i++){
            double prob = Math.pow(demand,i)*Math.exp(-1*demand)/factor...
...ial(i);

            probLTCapacity += prob;
```

```
        }

        double probGTCapacity = 1.0D - probLTCapacity;

        if(probGTCapacity < 0){
            return 0.0D;
        }else if(probGTCapacity > 1){
            return 1.0D;
        }else{
            return probGTCapacity;
        }


    }

    private double factorial(int n) {
        if(this.factorials_.size() == 0){
            factorials_.add(1.0D);
            for(int i=1; i<=100; i++){
                factorials_.add(i*factorials_.get(i-1));
            }
        }
        if (n > 100) {
            return Double.MAX_VALUE;
        } else {
            return factorials_.get(n);
        }
    }

    // Get list of flights operating in constrained sectors
    public ArrayList<String> getFlightsThroughFlowConstrainedArea(
            ArrayList<ASector> constrainedSectors){

        ArrayList<String> flights = new ArrayList<String>();

        Iterator<String> e = APData.airspace.getFlightIterator();
        while(e.hasNext()){
            String flightId = e.next();

            if(!flightLinks_.containsKey(flightId)) continue;

            ArrayList<Link3D> flightLinks = flightLinks_.get(flightId);...
...             for(Link3D link:flightLinks){
                 ASector linkSector = link.getParentSector();
                 if(constrainedSectors.contains(linkSector)){
                     flights.add(flightId);
                     break;
```

```
                }
            }
        }

        return flights;
    }

    public ConcurrentHashMap<String, HashMap<Integer, Double>> getFligh...
...tDemandByTimePeriod(){
        return flightDemandTimePeriod_;
    }


    // For weather scenarios remove the demand due to the constrained f...
...lights
    // to prepare for reassignment.
    public Scenario removeDemandFromConstrainedFlights(ArrayList<String...
...> constrainedFlights,
            ConcurrentHashMap<String, HashMap<Integer, Double>> flightD...
...emandTimePeriod){
        for(String flightId:constrainedFlights){
            if(!flightDemandTimePeriod.containsKey(flightId)) continue;...
...
            HashMap<Integer, Double> flightDemand = flightDemandTimePer...
...iod.get(flightId);

            for(int index:flightDemand.keySet()){
                int timePeriod = (int)(index % 10000);
                int sectorIndex = (int)((index - timePeriod)/10000);

                double demandToRemove = flightDemand.get(index);

                sectorFlowByMin_[sectorIndex][timePeriod] -= demandToRe...
...move;
            }
        }

        return this;
    }

    public double[] getConstrainedFlightLat(String flightId){
        ArrayList<PositionAtNode> flightPath = flightPath_.get(flightId...
...);

        double[] lats = new double[flightPath.size()];
        for(int k=0; k<flightPath.size(); k++){
            PositionAtNode pan = flightPath.get(k);
```

```
            Node3D<Double> node = pan.getNode();
            lats[k] = node.getLatitude();
        }

        return lats;
    }

    public double[] getConstrainedFlightLon(String flightId){
        ArrayList<PositionAtNode> flightPath = flightPath_.get(flightId...
...);

        double[] lons = new double[flightPath.size()];
        for(int k=0; k<flightPath.size(); k++){
            PositionAtNode pan = flightPath.get(k);
            Node3D<Double> node = pan.getNode();
            lons[k] = node.getLongitude();
        }

        return lons;
    }

    public Scenario setFlightStartClusterIndex(String flightId, int ind...
...ex){
        flightStartClusterIndex_.put(flightId, index);
        return this;
    }

    public int getFlightStartClusterIndex(String flightId){
        return flightStartClusterIndex_.get(flightId);
    }

    public Scenario setFlightEndClusterIndex(String flightId, int index...
...){
        flightEndClusterIndex_.put(flightId, index);
        return this;
    }

    public int getFlightEndClusterIndex(String flightId){
        return flightEndClusterIndex_.get(flightId);
    }

    /*
     *private HashMap<String, Integer> clusterIndexForFlight_;
    private HashMap<Integer, ArrayList<String>> flightsInCluster_;
    private HashMap<Integer, Node3D<Double>> clusterStartMedoid_;
    private HashMap<Integer, Node3D<Double>> clusterEndMedoid_;
     */
```

```
    public Scenario addClusterIndexToFlight(String flightId, int cluste...
...rIndex){
        clusterIndexForFlight_.put(flightId, clusterIndex);

        ArrayList<String> clusterFlights = flightsInCluster_.containsKe...
...y(clusterIndex) ?
            flightsInCluster_.get(clusterIndex) : new ArrayList<String>...
...();

        if(!clusterFlights.contains(flightId)) clusterFlights.add(fligh...
...tId);

        flightsInCluster_.put(clusterIndex, clusterFlights);

        return this;
    }

    public ArrayList<String> getFlightsInCluster(int clusterIndex){
        return flightsInCluster_.get(clusterIndex);
    }

    // Number of clusters
    public int noClusters(){
        return Settings.numberFlightClustersLT18kft +
                Settings.numberFlightClustersGT18kft;
    }

    public int getClusterIndexForFlight(String flightId){
        return clusterIndexForFlight_.get(flightId);
    }

    public Scenario addClusterStartMedoid(int clusterIndex, String flig...
...htId){
        int flIdx = flightStartClusterIndex_.get(flightId);
        PositionAtNode pan = getFlightPosition(flightId, flIdx);
        clusterStartMedoid_.put(clusterIndex, pan.getNode());

        return this;
    }

    public Node3D<Double> getClusterStartMedoid(int clusterIndex){
        return clusterStartMedoid_.get(clusterIndex);
    }

    public Scenario addClusterEndMedoid(int clusterIndex, String flight...
...Id){
```

```
        int flIdx = flightEndClusterIndex_.get(flightId);
        PositionAtNode pan = getFlightPosition(flightId, flIdx);
        clusterEndMedoid_.put(clusterIndex, pan.getNode());

        return this;
    }

    public Node3D<Double> getClusterEndMedoid(int clusterIndex){
        return clusterEndMedoid_.get(clusterIndex);
    }

    public HashMap<Node3D<Double>,Boolean> extractNetworkNodesForCluste...
...r(int clusterIndex){

        HashMap<Node3D<Double>,Boolean> nodesToReturn =
                new HashMap<Node3D<Double>,Boolean>(network_.noNodes())...
...;

        ArrayList<String> flightIdentifiers = flightsInCluster_.get(clu...
...sterIndex);

        double minFlightLevel = Double.MAX_VALUE;
        double maxFlightLevel = Double.MIN_VALUE;

        for(String flightId:flightIdentifiers){
            if(!flightPath_.containsKey(flightId)) continue;

            ArrayList<PositionAtNode> path = flightPath_.get(flightId);...
...            for(PositionAtNode pan:path){
                Node3D<Double> panNode = pan.getNode();
                if(!nodesToReturn.containsKey(panNode)) nodesToReturn.p...
...ut(panNode,true);
            }

            Flight f = airspace_.getFlight(flightId);
            double flLevel = f.getCruisingFlightLevel();

            minFlightLevel = Math.min(minFlightLevel, flLevel);
            maxFlightLevel = Math.max(maxFlightLevel, flLevel);
        }

        for(Node3D<Double> node: network_.getNodeAirportSet()){

            // Exclude nodes going through null sector
            //if(node.getParentSector().equals(ASector.NullSector)) con...
...tinue;
```

```
            double nodeFloor = node.getParentSector().getFloor();
            double nodeCeiling = node.getParentSector().getCeiling();

            boolean inRange = (minFlightLevel >= nodeFloor && minFlight...
...Level <= nodeCeiling);
            if(!inRange){
                inRange = (maxFlightLevel >= nodeFloor && maxFlightLeve...
...l <= nodeCeiling);
            }

            if(node.getElevation() >= minFlightLevel &&
                    node.getElevation() <= maxFlightLevel &&
                    !nodesToReturn.containsKey(node)){
                nodesToReturn.put(node,true);
            }else{
                nodesToReturn.put(node,false);
            }
        }

        //nodesToReturn.trimToSize();
        return nodesToReturn;


    }

    public HashMap<Link3D,Boolean> extractNetworkLinksForCluster(HashMa...
...p<Node3D<Double>,Boolean> nodes){

        HashMap<Link3D,Boolean> linksToReturn =
                new HashMap<Link3D,Boolean>(network_.noLinks());

        Iterator<String> e = network_.getLinkIterator();
        while(e.hasNext()){
            Link3D link = network_.getLink(e.next());

            // Exclude links going through null sector
            //if(link.getFromNode().getParentSector().equals(ASector.Nu...
...llSector) ||
            //        link.getToNode().getParentSector().equals(ASector...
....NullSector))
            //            continue;

            if(nodes.get(link.getFromNode()) && nodes.get(link.getToNod...
...e())){
                linksToReturn.put(link,true);
            }else{
                linksToReturn.put(link,false);
```

```
            }

            //if(nodes.contains(link.getFromNode()) &&
            //        nodes.contains(link.getToNode())) linksToReturn.a...
...dd(link);
        }

        //linksToReturn.trimToSize();
        return linksToReturn;
    }

    public HashMap<Link3D,Double> calculatePreliminaryLinkTravelTimeFor...
...Cluster(int clusterIndex,
            HashMap<Link3D,Boolean> links, double penaltyAltitudeOutOfR...
...ange){
        // If the timeToNode argument for calculatFinalLinkTravelTimeFo...
...rCluster is not
        // null then the preliminary result will be returned

        return calculateFinalLinkTravelTimeForCluster(clusterIndex,
            links, penaltyAltitudeOutOfRange, new HashMap<String,Double...
...>());
    }

    public HashMap<Link3D,Double> calculateFinalLinkTravelTimeForCluste...
...r(int clusterIndex,
            HashMap<Link3D,Boolean> links, double penaltyAltitudeOutOfR...
...ange,
            HashMap<String,Double> timeToNode){

        HashMap<Link3D,Double> linkTravelTime = new HashMap<Link3D,Doub...
...le>(links.size());

        ArrayList<String> flightIdentifiers = flightsInCluster_.get(clu...
...sterIndex);
        ArrayList<EAircraft> flightAcs = new ArrayList<EAircraft>(fligh...
...tIdentifiers.size());

        // Get flight departure times (this is an approximation - will ...
...be updated in future)
        ArrayList<Double> flightDeps = new ArrayList<Double>(flightIden...
...tifiers.size());

        // Variables for calculating average cruising flight level
        double totCruiseFl = 0.0D;
        int cruiseFlCnt = 0;
        for(String flightId:flightIdentifiers){
```

```
            Flight f = airspace_.getFlight(flightId);
            flightAcs.add(f.getAircraftType());
            flightDeps.add(f.getDepartureTimeMinutes());

            totCruiseFl += f.getCruisingFlightLevel();
            cruiseFlCnt++;
        }

        // Calculate average cruising flight level
        double averageCruiseFl = totCruiseFl/cruiseFlCnt;

        // Calculate average airspeed at altitude
        double totalAirspeed = 0.0D;
        for (EAircraft ac : flightAcs) {
            totalAirspeed += ac.getCruiseAirspeed(averageCruiseFl);
        }
        double averageAirspeed = totalAirspeed / flightAcs.size();

        int i;
        Set<Map.Entry<Link3D,Boolean>> set = links.entrySet();
        for(Map.Entry<Link3D,Boolean> entry:set){
            Link3D link = entry.getKey();
            boolean linkInRange = entry.getValue();

            // Calculate average wind speed and direction
            double totalWindSpeed = 0.0D;
            double sin_bar = 0.0D;
            double cos_bar = 0.0D;
            String fromNodeId = link.getFromNodeId();
            for(i=0; i<flightDeps.size(); i++){
                double time2 = timeToNode != null && timeToNode.contain...
...sKey(fromNodeId) ?
                    flightDeps.get(i) + timeToNode.get(fromNodeId) : fl...
...ightDeps.get(i);
                totalWindSpeed += link.getWindSpeed(time2, averageCruis...
...eFl);
                double dir = Math.toRadians(link.getWindDirection(time2...
..., averageCruiseFl));
                sin_bar += Math.sin(dir);
                cos_bar += Math.cos(dir);
            }
            sin_bar = sin_bar / cruiseFlCnt;
            cos_bar = cos_bar / cruiseFlCnt;
            double averageWindDirection = cos_bar < 0 ?
                Math.atan(sin_bar/cos_bar) + Math.PI : Math.atan(sin_ba...
...r/cos_bar);
            double averageWindSpeed = totalWindSpeed / cruiseFlCnt;
```

```
            double linkAzimuth = Math.toRadians(link.getAzimuth());

            // Find the angle (theta) between the link and the average ...
...wind direction
            double theta = Math.acos(Math.sin(averageWindDirection)*Mat...
...h.sin(linkAzimuth) +
                    Math.cos(averageWindDirection)*Math.cos(linkAzimuth...
...));

            double groundSpeed = Math.sqrt(Math.pow(averageAirspeed,2) ...
...+
                    Math.pow(averageWindSpeed,2)
                    - 2*averageAirspeed*averageWindSpeed*Math.cos(theta...
...));

            double travelTimeMinutes;
            if(Settings.useWindInFlightTimeCalculations){
                travelTimeMinutes = (link.getLength()/groundSpeed) * 60...
....0D;
            }else{
                travelTimeMinutes = (link.getLength()/averageAirspeed) ...
...* 60.0D;
            }

            // Add a penalty if the link is not in range
            if(!linkInRange) travelTimeMinutes *= penaltyAltitudeOutOfR...
...ange;

            linkTravelTime.put(link,travelTimeMinutes);
        }

        return linkTravelTime;
    }

    public void createDiGraphForCluster(int clusterIndex, double penalt...
...yAltitudeOutOfRange, int labelIndex){


        HashMap<Node3D<Double>,Boolean> nodes = extractNetworkNodesForC...
...luster(clusterIndex);
        HashMap<Link3D,Boolean> links = extractNetworkLinksForCluster(n...
...odes);
        HashMap<Link3D,Double> linkTravelTimes = calculatePreliminaryLi...
...nkTravelTimeForCluster(
                clusterIndex, links, penaltyAltitudeOutOfRange);
```

```
        for(Link3D link:linkTravelTimes.keySet()){
            double linkTravelTime = linkTravelTimes.get(link);

            link.setDoubleValue(labelIndex, linkTravelTime);

            Node3D<Double> fromNode = link.getFromNode();
            fromNode.setDoubleValue(labelIndex, 0.0D);
            Node3D<Double> toNode = link.getToNode();
            toNode.setDoubleValue(labelIndex, 0.0D);
        }
        // One-to-all shortest path
        DijkstraAdvanced spa = new DijkstraAdvanced(network_, labelInde...
...x, labelIndex);
        spa.valuateFromSource2(clusterStartMedoid_.get(clusterIndex));
        HashMap<String, Double> oneToAllResults = new HashMap<String, D...
...ouble>(network_.noNodes());
        Iterator<String> e = network_.getNodeIterator();
        while(e.hasNext()){
            String nodeId = e.next();
            Node3D<Double> node = network_.getNode(nodeId);
            double dist = spa.getDistanceTo(node);
            oneToAllResults.put(nodeId, dist);
        }

        linkTravelTimes = calculateFinalLinkTravelTimeForCluster(
                clusterIndex, links, penaltyAltitudeOutOfRange, oneToAl...
...lResults);

        // Set the revised travel time
        for(Link3D link:linkTravelTimes.keySet()){
            double linkTravelTime = linkTravelTimes.get(link);
            link.setDoubleValue(labelIndex, linkTravelTime);
        }



    }

    public static int numPathCalculations = 0;
    public static int numNullPaths = 0;

    public ArrayList<ArrayList<Link3D>> calculateKShortestPathsForClust...
...er(int clusterIndex, int numPathsToCalculate,
            double penaltyAltitudeOutOfRange, int labelIndex){


        //createDiGraphForCluster(clusterIndex, penaltyAltitudeOutOfRan...
```

```
...ge, labelIndex);

        // Get start/end vertex
        Node3D<Double> startNode = clusterStartMedoid_.get(clusterIndex...
...);
        Node3D<Double> endNode = clusterEndMedoid_.get(clusterIndex);

        ArrayList<ArrayList<Link3D>> kPaths = DijkstraAdvanced.KShortes...
...tPaths(
                numPathsToCalculate, startNode, endNode, labelIndex, la...
...belIndex, network_);

        // Use flight plan if shortest paths calculations fail
        numPathCalculations++;
        if(kPaths == null){
            numNullPaths++;
            kPaths = kPathsFlightPlan(clusterIndex, numPathsToCalculate...
...);
        }

        // Use flight plan as a fallback if the first path exceeds the ...
...flight
        // plan by 25% (length)
        boolean gtXPercent = pathGTFlightPlanXPercent(clusterIndex, kPa...
...ths);
        this.useFlightPlan_.put(clusterIndex, gtXPercent);

        return kPaths;
    }

    // Check if should use flight plan
    public boolean useFlightPlan(String flightId){
        int clusterIndex = this.clusterIndexForFlight_.get(flightId);
        return this.useFlightPlan_.get(clusterIndex);
    }

    // Check the length of the path compared to the flight plan
    public boolean pathGTFlightPlanXPercent(int clusterIndex,
            ArrayList<ArrayList<Link3D>> kPaths){

        // Calculate the length of the calculated path
        double pathLength = 0.0D;
        for(Link3D link:kPaths.get(0)){
            Node3D<Double> fromNode = link.getFromNode();
            Node3D<Double> toNode = link.getToNode();

            Mapping m = new Mapping(fromNode.getLatitude(),
```

```
                    fromNode.getLongitude(),
                    toNode.getLatitude(), toNode.getLongitude());

            pathLength += m.getDistance();
        }

        // Calculate the length of the flight plan path
        ArrayList<String> flightsInCluster = this.getFlightsInCluster(c...
...lusterIndex);
        String firstFlightId = flightsInCluster.get(0);
        Flight f = APData.airspace.getFlight(firstFlightId);
        FTrajectory t = f.getPlannedTrajectory();
        double planLength = 0.0D;

        // Get the length from the origin airport
        Mapping m = new Mapping(f.getOriginAirport().getLatitude(),
                f.getOriginAirport().getLongitude(),
                t.getWaypoint(0).getLatitude(),
                t.getWaypoint(0).getLongitude());
        planLength += m.getDistance();

        // Get the length between each of the waypoint
        for(int i=1; i<t.noWaypoints(); i++){
            m = new Mapping(t.getWaypoint(i-1).getLatitude(),
                    t.getWaypoint(i-1).getLongitude(),
                    t.getWaypoint(i).getLatitude(),
                    t.getWaypoint(i).getLongitude());
            planLength += m.getDistance();
        }

        // Get the length from the last waypoint to the destination air...
...port
        m = new Mapping(t.getLastWaypoint().getLatitude(),
                t.getLastWaypoint().getLongitude(),
                f.getDestinationAirport().getLatitude(),
                f.getDestinationAirport().getLongitude());
        planLength += m.getDistance();

        // Check the length
        boolean gtXPercent = pathLength >= 1.25 * planLength;

        return gtXPercent;
    }

    // Use the links from the flight plan if the k-shortest path calcul...
...ations fail
    public ArrayList<ArrayList<Link3D>> kPathsFlightPlan(int clusterInd...
```

```
...ex,
            int numPathsToCalculate){

        // Get the first flight in the cluster
        ArrayList<String> flightsInCluster = this.getFlightsInCluster(c...
...lusterIndex);

        // Check for null
        if(flightsInCluster == null) return null;

        String firstFlightId = flightsInCluster.get(0);

        // Get start and end index for the clustering
        int startIndex = flightStartClusterIndex_.get(firstFlightId);
        int endIndex = flightEndClusterIndex_.get(firstFlightId);

        // Get the path for the flight
        ArrayList<PositionAtNode> flightPath = flightPath_.get(firstFli...
...ghtId);
        PositionAtNode startPosition = flightPath.get(startIndex);
        PositionAtNode endPosition = flightPath.get(endIndex);

        Node3D<Double> startNode = startPosition.getNode();
        Node3D<Double> endNode = endPosition.getNode();

        // Get the planned links for this flight
        ArrayList<Link3D> flightLinks = flightLinks_.get(firstFlightId)...
...;

        ArrayList<Link3D> pathToReturn = new ArrayList<Link3D>();

        boolean foundStartNode = false;
        for(int i=0; i<flightLinks.size(); i++){
            Link3D link = flightLinks.get(i);

            if(foundStartNode || link.getFromNode().equals(startNode)){...
...                 foundStartNode = true;

                if(link.getFromNode().equals(endNode)){
                    break;
                }

                pathToReturn.add(link);
            }
        }

        // Create duplicate paths of length numPathsToCalculate
```

```
        ArrayList<ArrayList<Link3D>> pathsToReturn =
                new ArrayList<ArrayList<Link3D>>(numPathsToCalculate);
        for(int i=0; i<numPathsToCalculate; i++){
            pathsToReturn.add(pathToReturn);
        }

        return pathsToReturn;
    }

    // Thread class for k-shortest path algorithm calculations
    private static class kSpaCalculator implements Runnable{
        private Scenario scenario_;
        private int clusterIndex_;
        private int numPathsToCalculate_;
        private double penaltyAltitudeOutOfRange_;
        private int labelIndex_;

        public ArrayList<ArrayList<Link3D>> paths;

        public kSpaCalculator(Scenario scenario, int clusterIndex, int ...
...numPathsToCalculate,
                double penaltyAltitudeOutOfRange, int labelIndex){
            scenario_ = scenario;
            clusterIndex_ = clusterIndex;
            numPathsToCalculate_ = numPathsToCalculate;
            penaltyAltitudeOutOfRange_ = penaltyAltitudeOutOfRange;
            labelIndex_ = labelIndex;
        }

        public void run(){
            paths = scenario_.calculateKShortestPathsForCluster(cluster...
...Index_,
                    numPathsToCalculate_, penaltyAltitudeOutOfRange_, l...
...abelIndex_);
        }

    }

    public Scenario kShortestPathCalculations(int numPathsToCalculate, ...
...            double penaltyAltitudeOutOfRange){

        final int labelIndexLt18k = 10;
        final int labelIndexGt18k = 11;

        // Initialize the label values for the nodes and links
        network_.initializeNodeLinkDoubleValues(12);
```

```
        // Initialize graph for flights operating below 18kft
        // Use the first < 18kft cluster
        createDiGraphForCluster(1, penaltyAltitudeOutOfRange, labelInde...
...xLt18k);

        // Initialize graph for flights operating above 18kft
        // Use the first > 18kft cluster
        createDiGraphForCluster(Settings.numberFlightClustersLT18kft + ...
...1,
                penaltyAltitudeOutOfRange, labelIndexGt18k);

        // Remove links that are in sectors with zero capacity
        network_.removeLinksInSectorsWithZeroCapacity(labelIndexLt18k);...
...        network_.removeLinksInSectorsWithZeroCapacity(labelIndexGt18k);...
...
        // Collection of threads
        ArrayList<kSpaCalculator> calcs = new ArrayList<kSpaCalculator>...
...(noClusters());
        ArrayList<Thread> threads = new ArrayList<Thread>(noClusters())...
...;

        for(int i=0; i<noClusters(); i++){
            // Cluster index is 1 based
            int i2 = i+1;

            // Use modulus operator to get label index
            int labelIndex = i2 % 10;

            // Reset the label values
            network_.resetNodeLinkDoubleValues(labelIndex);

            if(i2 > Settings.numberFlightClustersLT18kft){
                network_.copyNodeLinkDoubleValues(labelIndexGt18k, labe...
...lIndex);
            }else{
                network_.copyNodeLinkDoubleValues(labelIndexLt18k, labe...
...lIndex);
            }

            kSpaCalculator kSpaCalc = new kSpaCalculator(this, i2,
                    numPathsToCalculate, penaltyAltitudeOutOfRange, lab...
...elIndex);
            Thread tKSpa = new Thread(kSpaCalc);

            // Add to collections
            calcs.add(kSpaCalc);
```

```
            threads.add(tKSpa);

            if(threads.size() <= 10){
                tKSpa.start();
            }else{
                // Wait for the previous label to become available
                Thread tKSpa2 = threads.get(i - 10);
                try{
                    tKSpa2.join();
                }catch(InterruptedException e){
                    System.out.println("Could not finish k-shortest pat...
...hs!");
                }

                tKSpa.start();
            }

//            ArrayList<ArrayList<Link3D>> paths = calculateKShortestPa...
...thsForCluster(i2,
//                    numPathsToCalculate, penaltyAltitudeOutOfRange, l...
...abelIndex);
//            kShortestPathsByCluster_.put(i2, paths);
//
//            System.out.println("    k shortest path for cluster: " + ...
...Integer.toString(i2));
//
            //if(i2 >= 50) break;
            //return this;
            if(i2 % 10 == 0){
                System.out.println("     k-Shortest paths for cluster: ...
..."
                        + Integer.toString(i2));
            }
        }
        for (int i = 0; i < calcs.size(); i++) {
            // Make sure thread is finished
            Thread tKSpa = threads.get(i);
            try {
                tKSpa.join();
            } catch (InterruptedException e) {
                System.out.println("Could not finish k-shortest paths!"...
...);
            }
            // Cluster index is 1 based
            int i2 = i + 1;
            kSpaCalculator kSpaCalc = calcs.get(i);
            // Store the k-shortest paths
```

```
            kShortestPathsByCluster_.put(i2, kSpaCalc.paths);
        }

        return this;
    }

    public ConcurrentHashMap<Integer, ArrayList<ArrayList<Link3D>>> KSh...
...ortestPathsByCluster(){
        return kShortestPathsByCluster_;
    }

    public ConcurrentHashMap<Integer, ArrayList<Link3D>> IncrementalPat...
...hsByCluster(){
        return incrementalPathsByCluster_;
    }

    public enum FlightStatus{ CLIMBING, CRUISING, DESCENDING }

    // Assign a flight to the network for this scenario
    // Returns true if no error, false if error in assignment
    public boolean assignFlightDemand(Flight f){
        // From node is the origin airport
        String fromNodeId = f.getOriginAirport().getAirspaceIdentifier(...
...) + APData._APT;
        Node3D<Double> fromNode = network_.getNode(fromNodeId);

        // Current traversal time distribution
        TraversalTimeDistribution ttd = new TraversalTimeDistribution(
                f.getOriginAirport().getAirportType());

        // Add from node to the flight path
        this.addFlightPosition(f.getCallSign(), fromNodeId, fromNode,
                f.getDepartureTimeMinutes(), 0, fromNode.getElevation()...
..., ttd);

        // Get node representing the destination airport
        String destAptNodeId = f.getDestinationAirport().getAirspaceIde...
...ntifier() + APData._APT;
        Node3D<Double> destAptNode = network_.getNode(destAptNodeId);
        // Keep track of current time, distance, and altitude
        double currentTime = f.getDepartureTimeMinutes();
        double currentDistance = 0.0D;
        double currentAltitude = f.getCruisingFlightLevel();

        // Flag to indicate stage of flight
        FlightStatus flightStatus = FlightStatus.CLIMBING;
```

```
        // Trajectory for the flight
        FTrajectory trajectory = f.getPlannedTrajectory();
        String previousTrajectoryPtId = APData.NULLSTRING;
        String trajectoryPtId = APData.NULLSTRING;
        Node3D<Double> toNode = null;
        for(int j=1; j <= trajectory.noWaypoints() + 1; j++){
            // Don't consider duplicate identifiers
            if(previousTrajectoryPtId.length() > 0 &&
                    j < trajectory.noWaypoints() + 1 &&
                    trajectory.getWaypointIdentifier(j-1).equals(previo...
...usTrajectoryPtId))
                continue;

            if(j < trajectory.noWaypoints() + 1){
                trajectoryPtId = trajectory.getWaypointIdentifier(j-1);...
...            }else if(j == trajectory.noWaypoints() + 1 &&
                    noFlightLinks(f.getCallSign()) == 0){
                currentAltitude = 999.0D;
                break;
            }else{
                // Need to go down to the lowest centroid for the last ...
...waypoint
                // before the airport.
                double tmpAltitude = currentAltitude;
                currentAltitude = 0.0D;

                // Maximum of 10 iterations
                for(int i=0; i < 5; i++){
                    int lastLinkIndex = noFlightLinks(f.getCallSign()) ...
...- 1;
                    Link3D lastLink = getFlightLink(f.getCallSign(), la...
...stLinkIndex);

                    double linkFloorAltitude = lastLink.getFloorAltitud...
...e();

                    // Recheck altitude for external to centroid links
                    ASector lastLinkSector = lastLink.getToNode().getPa...
...rentSector();
                    if(!lastLinkSector.equals(ASector.NullSector)){
                        String airspaceId = toNode.getAirspaceIdentifie...
...r();
                        if(lastLinkSector.hasFix(airspaceId)){
                            linkFloorAltitude = lastLinkSector.getFixMo...
...duleFloor(airspaceId);
                        }else if(lastLinkSector.hasNavaid(airspaceId)){...
...                             linkFloorAltitude = lastLinkSector.getNavai...
```

```
...dModuleFloor(airspaceId);
                        }
                    }

                    if(linkFloorAltitude < 1.0D){
                        fromNode = lastLink.getToNode();
                        break;
                    }

                    // Change to a different sector if necessary
                    ChangeAltitude changeAltitude = new ChangeAltitude(...
...                             currentTime, currentDistance, currentAltitu...
...de,
                            toNode, this, f);
                    changeAltitude.run(ttd);
                    toNode = changeAltitude.getToNode();

                }

                currentAltitude = tmpAltitude;
                trajectoryPtId = f.getDestinationAirport().getAirspaceI...
...dentifier();
            }

            previousTrajectoryPtId = trajectoryPtId;
            ArrayList<String> trajectoryPtNodeIds = network_.getNodesUs...
...ingIdentifier(trajectoryPtId);

            // Examine the links exiting the node
            ArrayList<Link3D> outLinks = fromNode.getOutLinkList();

            for(Link3D outLink:outLinks){
                toNode = outLink.getToNode();
                String toNodeId = outLink.getToNodeId();

                // Last node must be an airport
                if(j < trajectory.noWaypoints() + 1 && toNode.isAtAirpo...
...rt()){
                    continue;
                }else if(j == trajectory.noWaypoints() + 1 && !toNode.i...
...sAtAirport()){
                    continue;
                }

                // Check if need to go through boundary node to get to ...
...the next waypoint
                if(toNode.isAtBoundary() && fromNode.hasSameAirspaceIde...
```

```
...ntifier(toNode)){
                    ArrayList<Link3D> outLinks2 = toNode.getOutLinkList...
...();

                    // Next sector null
                    boolean nextSectorNull = false;

                    // Invalid to node airspace identifier
                    boolean toNodeAirspaceIdValid = true;

                    // Need to get the link that goes from a boundary n...
...ode to
                    // a boundary node
                    boolean hasCandidate = false;

                    Link3D boundaryLink = null;
                    for(Link3D candidateLink:outLinks2){
                        // Check that the airspaceId is valid
                        ASector candidateNextSector = candidateLink.get...
...ToNode().getParentSector();
                        String airspaceId = candidateLink.getToNode().g...
...etAirspaceIdentifier();
                        if(!airspaceId.equals(trajectoryPtId)){
                            toNodeAirspaceIdValid = false;
                            continue;
                        }else{
                            toNodeAirspaceIdValid = true;
                        }

                        // Consider case where boundary link goes to a ...
...null sector
                        if(candidateLink.getToNode().isAtCentroid() &&
                                trajectoryPtNodeIds.contains(candidateL...
...ink.getToNodeId())){
                            boundaryLink = candidateLink;
                            nextSectorNull = true;
                            break;
                        }

                        // Only consider links that go to another bound...
...ary node
                        if(!candidateLink.getToNode().isAtBoundary()) c...
...ontinue;

                        // Check altitude limits
                        double nextSectorFloor;
                        double nextSectorCeiling;
```

```
                        if(candidateNextSector.hasFix(airspaceId)){
                            nextSectorFloor = candidateNextSector.getFi...
...xModuleFloor(airspaceId);
                            nextSectorCeiling = candidateNextSector.get...
...FixModuleCeiling(airspaceId);
                        }else if(candidateNextSector.hasNavaid(airspace...
...Id)){
                            nextSectorFloor = candidateNextSector.getNa...
...vaidModuleFloor(airspaceId);
                            nextSectorCeiling = candidateNextSector.get...
...NavaidModuleCeiling(airspaceId);
                        }else{
                            continue;
                        }

                        // Set boundary link if within altitude limits
                        if(currentAltitude >= nextSectorFloor &&
                                currentAltitude <= nextSectorCeiling){
                            boundaryLink = candidateLink;
                            break;
                        }else if(currentAltitude < 180.0D && currentAlt...
...itude >= candidateLink.getFloorAltitude()
                                && currentAltitude <= nextSectorCeiling...
...){
                            boundaryLink = candidateLink;
                            hasCandidate = true;
                        }

                        // Default
                        if(!hasCandidate) boundaryLink = candidateLink;...
...                    }

                    // Make sure boundary link has been set
                    if(boundaryLink == null) continue;

                    ArrayList<Link3D> outLinks3 = boundaryLink.getToNod...
...e().getOutLinkList();

                    boolean nextSectorLinkSet = false;

                    // Need to get the link that goes from a boundary n...
...ode to a centroid node
                    Link3D nextSectorLink = null;
                    if(nextSectorNull){
                        nextSectorLinkSet = true;
                        nextSectorLink = boundaryLink;
                    }else{
```

```
                        for(Link3D candidateLink:outLinks3){
                            // Only consider links that go to a centroi...
...d node from
                            // a boundary node.
                            if(!candidateLink.getToNode().isAtCentroid(...
...) ||
                                    !trajectoryPtNodeIds.contains(candi...
...dateLink.getToNodeId()))
                                continue;

                            if(!nextSectorLinkSet){
                                nextSectorLink = candidateLink;
                                nextSectorLinkSet = true;
                            }

                            if(currentAltitude >= candidateLink.getFloo...
...rAltitude() &&
                                    currentAltitude <= candidateLink.ge...
...tCeilingAltitude()){
                                nextSectorLink = candidateLink;
                                break;
                            }
                        }
                    }

                    //if(nextSectorLinkSet && currentAltitude >= nextSe...
...ctorLink.getFloorAltitude() &&
                    //        currentAltitude <= nextSectorLink.getCeil...
...ingAltitude() &&
                    //        trajectoryPtNodeIds.contains(nextSectorLi...
...nk.getToNodeId())){
                    if(nextSectorLinkSet && trajectoryPtNodeIds.contain...
...s(nextSectorLink.getToNodeId())){
                        fromNode = nextSectorLink.getFromNode();
                        toNode = nextSectorLink.getToNode();
                        toNodeId = nextSectorLink.getToNodeId();

                        // Record the results
                        UpdateTimeAndAltitude updater = new UpdateTimeA...
...ndAltitude(
                                currentTime, currentDistance, currentAl...
...titude,
                                outLink, f.getAircraftType(), f.getCrui...
...singFlightLevel(),
                                flightStatus, destAptNode);
                        updater.run();
                        currentTime = updater.getCurrentTime();
```

```
                        currentDistance = updater.getCurrentDistance();...
...                        currentAltitude = updater.getCurrentAltitude();...
...                        flightStatus = updater.getFlightStatus();
                        double previousTime = updater.getPreviousTime()...
...;

                        addFlightPosition(f.getCallSign(), outLink.getT...
...oNodeId(),
                                outLink.getToNode(), currentTime, curre...
...ntDistance, currentAltitude,
                                ttd);
                        addFlightLink(f.getCallSign(), outLink.getIdent...
...ifier(), outLink);

                        if(!nextSectorNull){
                            outLink = boundaryLink;

                            updater = new UpdateTimeAndAltitude(
                                    currentTime, currentDistance, curre...
...ntAltitude,
                                    outLink, f.getAircraftType(), f.get...
...CruisingFlightLevel(),
                                    flightStatus, destAptNode);
                            updater.run();
                            currentTime = updater.getCurrentTime();
                            currentDistance = updater.getCurrentDistanc...
...e();
                            currentAltitude = updater.getCurrentAltitud...
...e();
                            flightStatus = updater.getFlightStatus();
                            previousTime = updater.getPreviousTime();

                            addFlightPosition(f.getCallSign(), outLink....
...getToNodeId(),
                                    outLink.getToNode(), currentTime, c...
...urrentDistance, currentAltitude,
                                    ttd);
                            addFlightLink(f.getCallSign(), outLink.getI...
...dentifier(), outLink);
                        }

                        outLink = nextSectorLink;
                        updater = new UpdateTimeAndAltitude(
                                currentTime, currentDistance, currentAl...
...titude,
                                outLink, f.getAircraftType(), f.getCrui...
...singFlightLevel(),
```

```
                            flightStatus, destAptNode);
                    updater.run();
                    currentTime = updater.getCurrentTime();
                    currentDistance = updater.getCurrentDistance();...
...                    currentAltitude = updater.getCurrentAltitude();...
...                    flightStatus = updater.getFlightStatus();
                    previousTime = updater.getPreviousTime();

                    addFlightPosition(f.getCallSign(), outLink.getT...
...oNodeId(),
                            outLink.getToNode(), currentTime, curre...
...ntDistance, currentAltitude,
                            ttd);
                    addFlightLink(f.getCallSign(), outLink.getIdent...
...ifier(), outLink);
                    // Change to a different sector if necessary
                    if(currentAltitude < nextSectorLink.getFloorAlt...
...itude() ||
                            currentAltitude > nextSectorLink.getCei...
...lingAltitude()){
                        ChangeAltitude changeAltitude = new ChangeA...
...ltitude(
                                currentTime, currentDistance, curre...
...ntAltitude,
                                toNode, this, f);
                        changeAltitude.run(ttd);
                        toNode = changeAltitude.getToNode();
                    }

                    break;
                }
            }

            // Case from an external node to a boundary node for ne...
...xt waypoint
            if(trajectoryPtNodeIds.contains(toNodeId) && toNode.isA...
...tBoundary()){
                ArrayList<Link3D> outLinks3 = toNode.getOutLinkList...
...();
                boolean nextSectorLinkSet = false;

                Link3D nextSectorLink = null;
                for(Link3D candidateLink : outLinks3){

                    // Only consider links that go to a centroid no...
...de from a boundary node
                    if(!candidateLink.getToNode().isAtCentroid() ||...
```

```
...                                !candidateLink.getToNode().hasSameAirsp...
...aceIdentifier(toNode))
                            continue;

                        nextSectorLink = candidateLink;
                        nextSectorLinkSet = true;
                    }

                    if(nextSectorLinkSet && currentAltitude >= nextSect...
...orLink.getFloorAltitude() &&
                            currentAltitude <= nextSectorLink.getCeilin...
...gAltitude() &&
                            trajectoryPtNodeIds.contains(nextSectorLink...
....getToNodeId())){
                        fromNode = nextSectorLink.getFromNode();
                        toNode = nextSectorLink.getToNode();
                        toNodeId = nextSectorLink.getToNodeId();

                        // Record the results
                        UpdateTimeAndAltitude updater = new UpdateTimeA...
...ndAltitude(
                                currentTime, currentDistance, currentAl...
...titude,
                                outLink, f.getAircraftType(), f.getCrui...
...singFlightLevel(),
                                flightStatus, destAptNode);
                        updater.run();
                        currentTime = updater.getCurrentTime();
                        currentDistance = updater.getCurrentDistance();...
...                        currentAltitude = updater.getCurrentAltitude();...
...                        flightStatus = updater.getFlightStatus();
                        double previousTime = updater.getPreviousTime()...
...;

                        addFlightPosition(f.getCallSign(), outLink.getT...
...oNodeId(),
                                outLink.getToNode(), currentTime, curre...
...ntDistance, currentAltitude,
                                ttd);
                        addFlightLink(f.getCallSign(), outLink.getIdent...
...ifier(), outLink);

                        outLink = nextSectorLink;
                        updater = new UpdateTimeAndAltitude(
                                currentTime, currentDistance, currentAl...
...titude,
                                outLink, f.getAircraftType(), f.getCrui...
```

```
...singFlightLevel(),
                                flightStatus, destAptNode);
                        updater.run();
                        currentTime = updater.getCurrentTime();
                        currentDistance = updater.getCurrentDistance();...
...                        currentAltitude = updater.getCurrentAltitude();...
...                        flightStatus = updater.getFlightStatus();
                        previousTime = updater.getPreviousTime();

                        addFlightPosition(f.getCallSign(), outLink.getT...
...oNodeId(),
                                outLink.getToNode(), currentTime, curre...
...ntDistance, currentAltitude,
                                ttd);
                        addFlightLink(f.getCallSign(), outLink.getIdent...
...ifier(), outLink);

                        break;
                    }
                }

                // Case where to node is the next point on the trajecto...
...ry
                if(trajectoryPtNodeIds.contains(toNodeId) && !toNode.is...
...AtBoundary() &&
                        !toNode.isAtFloor() && !toNode.isAtCeiling()){
                    UpdateTimeAndAltitude updater = new UpdateTimeAndAl...
...titude(
                            currentTime, currentDistance, currentAltitu...
...de,
                            outLink, f.getAircraftType(), f.getCruising...
...FlightLevel(),
                            flightStatus, destAptNode);
                    updater.run();
                    currentTime = updater.getCurrentTime();
                    currentDistance = updater.getCurrentDistance();
                    currentAltitude = updater.getCurrentAltitude();
                    flightStatus = updater.getFlightStatus();
                    double previousTime = updater.getPreviousTime();

                    addFlightPosition(f.getCallSign(), outLink.getToNod...
...eId(),
                            outLink.getToNode(), currentTime, currentDi...
...stance, currentAltitude,
                            ttd);
                    addFlightLink(f.getCallSign(), outLink.getIdentifie...
...r(), outLink);
```

```
                    break;
                }
            }

            // Check if should increase/decrease altitude at node to th...
...e next sector
            if(noFlightLinks(f.getCallSign()) == 0) continue;   // No c...
...heck if no link assigned
            Link3D outLink = getLastLink(f.getCallSign());
            double linkCeilingAltitude = getLastLinkCeilingAltitude(f.g...
...etCallSign());
            double linkFloorAltitude = getLastLinkFloorAltitude(f.getCa...
...llSign());

            // Get to node altitude in next sector
            if(linkFloorAltitude > 0){
                ASector nextSec = outLink.getToNode().getParentSector()...
...;
                String airspaceId = outLink.getToNode().getAirspaceIden...
...tifier();
                if(nextSec.hasFix(airspaceId)){
                    linkFloorAltitude = Math.max(nextSec.getFixModuleFl...
...oor(airspaceId),
                            linkFloorAltitude);
                }else if(nextSec.hasNavaid(airspaceId)){
                    linkFloorAltitude = Math.max(nextSec.getNavaidModul...
...eFloor(airspaceId),
                            linkFloorAltitude);
                }
            }

            // Make sure aircraft is at proper altitude
            int iterationCheck = 0;
            while(j < trajectory.noWaypoints() + 1 && currentAltitude >...
... 0 &&
                    (currentAltitude > linkCeilingAltitude || currentAl...
...titude < linkFloorAltitude)){

                // Call script to change altitude
                ChangeAltitude changeAltitude = new ChangeAltitude(
                        currentTime, currentDistance, currentAltitude,
                        toNode, this, f);
                changeAltitude.run(ttd);
                toNode = changeAltitude.getToNode();

                outLink = getLastLink(f.getCallSign());
```

```
                linkCeilingAltitude = getLastLinkCeilingAltitude(f.getC...
...allSign());
                linkFloorAltitude = getLastLinkFloorAltitude(f.getCallS...
...ign());

                addFlightPosition(f.getCallSign(), outLink.getToNodeId(...
...),
                        outLink.getToNode(), currentTime, currentDistan...
...ce, currentAltitude,
                        ttd);
                addFlightLink(f.getCallSign(), outLink.getIdentifier(),...
... outLink);

                // Check if too many iterations
                if(iterationCheck++ > 3) break;
            }

            // Reset the from node for the next iteration
            fromNode = outLink.getToNode();

        }

        // Error if no links assigned
        if(noFlightLinks(f.getCallSign()) == 0) return false;

        Link3D lastLink = getLastLink(f.getCallSign());

        if(!lastLink.getToNode().equals(destAptNode)){
//             if(firstError){
//                 // Print some diagnostics
//                 System.out.println("Error assigning flight : " + f.ge...
...tCallSign());
//
//                 // Origin airport
//                 System.out.println("Origin airport : " + f.getOriginA...
...irport().getAirspaceIdentifier());
//
//                 // Trajectory
//                 System.out.println("Trajectory info : ");
//                 for(int i=0; i<trajectory.noWaypoints(); i++){
//                     System.out.println(trajectory.getWaypointIdentifi...
...er(i));
//                 }
//
//                 // Destination airport
//                 System.out.println("Destination airport : " + f.getDe...
...stinationAirport().getAirspaceIdentifier());
```

```
//
//                // List of assigned nodes
//                System.out.println("List of assigned nodes : ");
//                Node3D<Double> lastNode = null;
//                for(PositionAtNode pan:flightPath_.get(f.getCallSign(...
...))){
//                    System.out.println(pan.getNode().getAirspaceIdent...
...ifier()
//                    + " : " + pan.getNode().getElevation().toString()...
...//                       + " : " + pan.getNode().getNodeType().toString())...
...;
//                    lastNode = pan.getNode();
//                }
//
//                // List of links exiting last node
//                System.out.println("List of nodes exiting last node :...
... ");
//                for(Link3D link:lastNode.getOutLinkList()){
//                    System.out.println(link.getToNode().getAirspaceId...
...entifier()
//                    + " : " + link.getToNode().getElevation() + " : "...
... +
//                            link.getToNode().getNodeType().toString()...
...);
//                }
//
//                firstError = false;
//            }

            // Error in assignment
            return false;
        }else{
            // OK
            return true;
        }
    }

//    private static boolean firstError = true;

    public Link3D getLastLink(String flightId){
        int lastLinkIndex = noFlightLinks(flightId) - 1;
        Link3D lastLink = getFlightLink(flightId, lastLinkIndex);
        return lastLink;
    }

    public double getLastLinkCeilingAltitude(String flightId){
        int lastLinkIndex = noFlightLinks(flightId) - 1;
```

```
        Link3D lastLink = getFlightLink(flightId, lastLinkIndex);
        double lastLinkCeilingAltitude = lastLink.getCeilingAltitude();...
...        return lastLinkCeilingAltitude;
    }

    public double getLastLinkFloorAltitude(String flightId){
        int lastLinkIndex = noFlightLinks(flightId) - 1;
        Link3D lastLink = getFlightLink(flightId, lastLinkIndex);
        double lastLinkFloorAltitude = lastLink.getFloorAltitude();
        return lastLinkFloorAltitude;
    }

    public double getTotalCapacity(){
        double totCapacity = 0.0D;
        int i;
        int j;
        for(i=0; i<sectorCapacity_.length; i++){
            double[] cap = sectorCapacity_[i];
            for(j=0; j<cap.length; j++){
                totCapacity += cap[j];
            }
        }
        return totCapacity;
    }

    /** Implementation of the Comparable interface for sorting */
    public int compareTo(Object obj) throws ClassCastException{
        Scenario wxScenario = (Scenario)obj;
        double thisCapacity = getTotalCapacity();
        double obsCapacity = wxScenario.getTotalCapacity();

        if(thisCapacity < obsCapacity) return -1;
        else if(thisCapacity > obsCapacity) return 1;
        else return 0;
    }


    /** Add the path obtained from incremental assignment to the scenar...
...io */
    public Scenario addIncrementalPathByCluster(int clusterIndex, Array...
...List<Link3D> path){
        incrementalPathsByCluster_.put(clusterIndex, path);

        return this;
    }

    /** Add demand for incremental assignment.
```

```
        This should be called from flightPlanScenario.
     */
    public Scenario incrementDemandFromIncrementalAssignment(
            int clusterIndex, ArrayList<Link3D> path,
            ArrayList<Scenario> wxScenarios, int labelIndex){

        // Store the result
        for(Scenario wxScenario:wxScenarios){
            wxScenario.addIncrementalPathByCluster(clusterIndex, path);...
...        }

        // Get the list of flights in the cluster
        ArrayList<String> flightsInCluster = this.getFlightsInCluster(c...
...lusterIndex);

        // Iterate through each of the flights
        int i;
        int j;
        int k;
        long i2;
        for(i=0; i<flightsInCluster.size(); i++){
            // Get the flight
            String flightId = flightsInCluster.get(i);
            Flight f = airspace_.getFlight(flightId);

            // Get the density for the flight (no flight has departed a...
...t this stage)
            KernelDensity kernelDensity = APData.kernelDensities.get(
                    f.getOriginAirport().getAirportType());
            ArrayList<Double> errorX = kernelDensity.getValues();
            ArrayList<Double> density = kernelDensity.getDensities();

            // Get the departure time
            double currentTime = f.getDepartureTimeMinutes();
            // Iterate through each link in the path
            for(j=0; j<path.size(); j++){
                Link3D link = path.get(j);


                // Update the time
                double previousTime = currentTime;
                if(link.getDoubleValue(labelIndex) == Double.MAX_VALUE)...
...{
                    currentTime = previousTime;
                }else{
                    currentTime = previousTime + link.getDoubleValue(la...
...belIndex);
```

```
                }


                // Get the sector
                ASector sector = link.getParentSector();
                if(sector.equals(ASector.NullSector)) continue;

                long previousTimeL = Math.round(previousTime);
                long currentTimeL = Math.round(currentTime);

                for(i2 = previousTimeL; i2 <= currentTimeL; i2++){
                    for(k=0; k<errorX.size(); k++){
                        double timeToAdd = i2 + errorX.get(k);
                        double valToAdd = density.get(k);

                        // Add the demand to each of the weather scenar...
...ios
                        for(Scenario wxScenario:wxScenarios){
                            wxScenario.incrementSectorFlow(flightId, se...
...ctor,
                                    timeToAdd, valToAdd);
                        }
                    }
                }
            }
        }

        return this;
    }

    // Add demand to the sector for incremental assignment
    public Scenario incrementSectorFlowIncremental(String flightId, ASe...
...ctor sector,
            double time, double valToAdd){
        int timeIndex = this.convertTimeToTimePeriodByMin(time);

        if(timeIndex >= Settings.numAnalysisTimePeriodsOneMin - 1
                || timeIndex < 0) return this;

        int sectorIndex = sector.getArrayIndex();

        if(sectorIndex < 0) return this;

        sectorFlowByMin_[sectorIndex][timeIndex] += valToAdd;

        return this;
    }
```

```
}
```

SectorTraversal.java

```
/*
 * SectorTraversal.java
 *
 * Created on September 3, 2007, 12:31 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author jeff
 */
import java.util.*;
public class SectorTraversal {
    public static final int NUM_MINUTES_PER_DAY = 1440;

    public String SectorId;

    /** Area of the sector in nm2 */
    public double Area;

    /** Volume of the sector in nm3 */
    public double Volume;

    public ArrayList<Double> PlannedDemandByMin;
    public ArrayList<Double> PlannedTotalTraversalTimeByMin;
    public ArrayList<Double> PlannedNumTraversalsByMin;
    public ArrayList<Double> PlannedAverageTraversalTimeByMin;
    public ArrayList<Double> PlannedDensityByMin2D;
    public ArrayList<Double> PlannedDensityByMin3D;

    public ArrayList<Double> DeterministicDemand10minLAT;
    public ArrayList<Double> DeterministicDemand30minLAT;
    public ArrayList<Double> DeterministicDemand60minLAT;

    public ArrayList<Double> ProbabilisticDemand10minLAT;
    public ArrayList<Double> ProbabilisticDemand30minLAT;
    public ArrayList<Double> ProbabilisticDemand60minLAT;

    public ArrayList<Double> ObservedDemandByMin;
    public ArrayList<Double> ObservedTotalTraversalTimeByMin;
    public ArrayList<Double> ObservedNumTraversalsByMin;
    public ArrayList<Double> ObservedAverageTraversalTimeByMin;
    public ArrayList<Double> ObservedDensityByMin2D;
    public ArrayList<Double> ObservedDensityByMin3D;
```

```
    /** Creates a new instance of SectorTraversal */
    public SectorTraversal(String sectorId) {
        SectorId = sectorId;

        PlannedDemandByMin = new ArrayList<Double>(NUM_MINUTES_PER_DAY)...
...;
        PlannedTotalTraversalTimeByMin = new ArrayList<Double>(NUM_MINU...
...TES_PER_DAY);
        PlannedNumTraversalsByMin = new ArrayList<Double>(NUM_MINUTES_P...
...ER_DAY);
        PlannedAverageTraversalTimeByMin = new ArrayList<Double>(NUM_MI...
...NUTES_PER_DAY);
        PlannedDensityByMin2D = new ArrayList<Double>(NUM_MINUTES_PER_D...
...AY);
        PlannedDensityByMin3D = new ArrayList<Double>(NUM_MINUTES_PER_D...
...AY);

        DeterministicDemand10minLAT = new ArrayList<Double>(NUM_MINUTES...
..._PER_DAY);
        DeterministicDemand30minLAT = new ArrayList<Double>(NUM_MINUTES...
..._PER_DAY);
        DeterministicDemand60minLAT = new ArrayList<Double>(NUM_MINUTES...
..._PER_DAY);

        ProbabilisticDemand10minLAT = new ArrayList<Double>(NUM_MINUTES...
..._PER_DAY);
        ProbabilisticDemand30minLAT = new ArrayList<Double>(NUM_MINUTES...
..._PER_DAY);
        ProbabilisticDemand60minLAT = new ArrayList<Double>(NUM_MINUTES...
..._PER_DAY);

        ObservedDemandByMin = new ArrayList<Double>(NUM_MINUTES_PER_DAY...
...);
        ObservedTotalTraversalTimeByMin = new ArrayList<Double>(NUM_MIN...
...UTES_PER_DAY);
        ObservedNumTraversalsByMin = new ArrayList<Double>(NUM_MINUTES_...
...PER_DAY);
        ObservedAverageTraversalTimeByMin = new ArrayList<Double>(NUM_M...
...INUTES_PER_DAY);
        ObservedDensityByMin2D = new ArrayList<Double>(NUM_MINUTES_PER_...
...DAY);
        ObservedDensityByMin3D = new ArrayList<Double>(NUM_MINUTES_PER_...
...DAY);

        for(int i=0; i < NUM_MINUTES_PER_DAY; i++){
            PlannedDemandByMin.add(0.0D);
```

```
            PlannedTotalTraversalTimeByMin.add(0.0D);
            PlannedNumTraversalsByMin.add(0.0D);
            PlannedAverageTraversalTimeByMin.add(0.0D);
            PlannedDensityByMin2D.add(0.0D);
            PlannedDensityByMin3D.add(0.0D);

            DeterministicDemand10minLAT.add(0.0D);
            DeterministicDemand30minLAT.add(0.0D);
            DeterministicDemand60minLAT.add(0.0D);

            ProbabilisticDemand10minLAT.add(0.0D);
            ProbabilisticDemand30minLAT.add(0.0D);
            ProbabilisticDemand60minLAT.add(0.0D);

            ObservedDemandByMin.add(0.0D);
            ObservedTotalTraversalTimeByMin.add(0.0D);
            ObservedNumTraversalsByMin.add(0.0D);
            ObservedAverageTraversalTimeByMin.add(0.0D);
            ObservedDensityByMin2D.add(0.0D);
            ObservedDensityByMin3D.add(0.0D);
        }

    }

    public double getPlannedTraversalTime(){
        double total = 0.0D;
        double count = 0.0D;
        for(int i=0; i < NUM_MINUTES_PER_DAY; i++){
            total += PlannedTotalTraversalTimeByMin.get(i);
            count += PlannedNumTraversalsByMin.get(i);
        }

        if(count == 0.0D) return 0.0D;

        return total/count;

    }

    public double getObservedTraversalTime(){
        double total = 0.0D;
        double count = 0.0D;
        for(int i=0; i < NUM_MINUTES_PER_DAY; i++){
            total += ObservedTotalTraversalTimeByMin.get(i);
            count += ObservedNumTraversalsByMin.get(i);
        }

        if(count == 0.0D) return 0.0D;
```

```
        return total/count;
    }

    public SectorTraversal calculateDensityByMin(){
        for(int i=0; i < NUM_MINUTES_PER_DAY; i++){
            PlannedDensityByMin2D.set(i, PlannedDemandByMin.get(i)/Area...
...);
            PlannedDensityByMin3D.set(i, PlannedDemandByMin.get(i)/Volu...
...me);

            ObservedDensityByMin2D.set(i, ObservedDemandByMin.get(i)/Ar...
...ea);
            ObservedDensityByMin3D.set(i, ObservedDemandByMin.get(i)/Vo...
...lume);
        }

        return this;
    }

    public SectorTraversal addDeterministicDemand(int lookAheadTime,
            int currentCorrectedEntry, int currentCorrectedExit){

        ArrayList<Double> DeterministicDemand = null;

        if(lookAheadTime == 10){
            DeterministicDemand = DeterministicDemand10minLAT;
        }else if(lookAheadTime == 30){
            DeterministicDemand = DeterministicDemand30minLAT;
        }else if(lookAheadTime == 60){
            DeterministicDemand = DeterministicDemand60minLAT;
        }

        for(int i=currentCorrectedEntry; i<=currentCorrectedExit; i++){...
...             if(i < 0 || i > 1439){
                 continue;
            }

            double currentDemand = DeterministicDemand.get(i);
            currentDemand += 1.0D;
            DeterministicDemand.set(i,currentDemand);
        }
        return this;
    }

    public SectorTraversal addProbabilisticDemand(int lookAheadTime,
            double currentCorrectedEntry, double[] xc,
```

```
            double[] errorDistribution, String flightId){

        ArrayList<Double> ProbabilisticDemand = null;

        if(lookAheadTime == 10){
            ProbabilisticDemand = ProbabilisticDemand10minLAT;
        }else if(lookAheadTime == 30){
            ProbabilisticDemand = ProbabilisticDemand30minLAT;
        }else if(lookAheadTime == 60){
            ProbabilisticDemand = ProbabilisticDemand60minLAT;
        }

        double total = 0;
        double outsideLimits = 0.0D;
        for(int i=0; i<xc.length; i++){
            double c2 = Math.round(currentCorrectedEntry + xc[i]);
            if(c2 < 0 || c2 > 1439){
                outsideLimits += errorDistribution[i];
                continue;
            }

            int idx = (int)c2;
            double currentDemand = ProbabilisticDemand.get(idx);
            currentDemand = currentDemand + errorDistribution[i];
            ProbabilisticDemand.set(idx, currentDemand);

            total += errorDistribution[i];
        }

        // xc goes from -90:0.5:120
        // Revision: don't do this - it degrades the estimation
//        int zeroIdx = 180;
//        double currentDemand = ProbabilisticDemand.get(zeroIdx);
//        currentDemand = currentDemand + outsideLimits;
//        ProbabilisticDemand.set(zeroIdx, currentDemand);

        if(total > 100.0D){
            System.out.println(flightId + " : " + this.SectorId + " : "...
... +
                    Integer.toString(lookAheadTime));
        }

        return this;

    }

    public double totalObservedDemand(){
```

```
        double total = 0.0D;
        for(int i=0; i<this.ObservedDemandByMin.size(); i++){
            total += ObservedDemandByMin.get(i);
        }
        return total;
    }

}
```

Settings.java

```
/*
 * Settings.java
 *
 * Created on August 6, 2006, 4:30 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
/**
 *
 * @author jeff
 */
import java.io.*;
import java.util.*;
public class Settings {
    public static String appDir;
    public static String dataDir;
    public static String dependentLibFile;
    public static String libFile;
    public static String errLogFile;
    public static String sectorFileName;
    public static String sectorMapFileName;
    public static String navaidSectorFileName;
    public static String fixSectorFileName;
    public static String airwaySectorIntersectionFileName;
    public static String airportFileName;
    public static String airportTypeFileName;
    public static String aircraftTypeFileName;
    public static String aircraftPreprocessPerf;
    public static String aircraftClimbDescentFile;
    public static String navaidFileName;
    public static String fixFileName;
    public static String natfixFile;
    public static String internationalWaypointsFile;
    public static String airwayFileName;
    public static String internationalAwyFile;
    public static String ramsFilesDir;
    public static String ramsOutFilesDir;
    public static String etmsFile;
    public static String etmsRadarTrackFile;
    public static String etmsFlightPlanFile;
    public static String etmsArrivalFile;
    public static String etmsDepartureFile;
    public static String outRadarTrackFile;
    public static String outRadarCoordinates;
    public static String outFlightIndex;
```

```
public static String outFlightIndexCallsign;
public static String outAltitudeStats;
public static String outJetRouteFlights;
public static String badaDirectory;
public static String windDirectory;
public static String windStations;
public static String windData;
public static String windNavaidClosestStation;
public static String windFixClosestStation;
public static String windNatfixClosestStation;
public static String starFile;
public static String j6FlightPlanFile;
public static String ncwfFile;
public static String kmlFile;
public static String linkLengthFile;
public static String convectionForecastDirectory;
public static String convectionTopsFilename;
public static String ncwfWeibullParametersFilename;
public static String kernelDensityFileName;
public static String sectorsToExclude;
public static String logFile;
public static String apcdmFile;
public static String apcdmFlightsInClusterFile;

/** MATLAB library file */
public static String matlabLibraryFile;
public static String matlabJniLibraryFile;

/** Only load the radar track information into the data structure.
 * Automatically overrides writeDummyAirwayFlights and
 * readDummyAirwayFlightsIntersections
 */
public static boolean readRadarTrackDataOnly;

/** Polygon used to filter flight track data */
public static int[] filterLongitude = {-105, -105, -63, -63};
public static int[] filterLatitude = {47, 25, 25, 47};
public static double filterMinAltitude;
public static double filterMaxAltitude;

/** Time filter for flight track data */
public static int filterMinTime;  // Minimum time in hours
public static int filterMaxTime;   // Maximum time in hours

public static boolean filterTime;
public static boolean filterAltitude;
```

```
    /** Number of lines of flight plan file to read */
    public static int flightPlanFileLinesToRead = Integer.MAX_VALUE;  /...
.../500;

    /** Number of lines of radar track file to read */
    public static int radarTrackFileLinesToRead = Integer.MAX_VALUE;  /...
.../1000;


    /** Number of time periods to analyze congestion.
        96 time periods (15 minutes) is 24 hours
     */
    public static int numAnalysisTimePeriods = 96;

    /** Number of time periods in 1 minute intervals to get maximum MAP...
... value
     *  during 15 minutes
     */
    public static int numAnalysisTimePeriodsOneMin = Settings.numAnalys...
...isTimePeriods*15;


    /** HashMap load factor. Default = 0.75
     *  Hashmap load factor (0.0->1.0).
        Closer to 0.0: Faster but higher memory requirements.
        Closer to 1.0: Slower and lower memory requirements.
     */
    public static final float hashMapLoadFactor = 0.70F;
    public static final int hashMapDefaultInitialCapacity = 64;


    /** Flow constrained sectors settings
     *  capacityProbabilityThreshold: probability of exceeding capacity...
...     *  proportionTimeCongested: percentage of time demand exceeding pr...
...obability
     *  threshold
     */
    public static final double capacityProbabilityThreshold = 0.55D; //...
... 10%
    public static final double proportionTimeCongested = 0.55D; // 30%

    /** Settings for flight clustering
     *
     */
    public static int numberFlightClustersLT18kft = 20;
    public static int numberFlightClustersGT18kft = 30;
```

```
    /* Arrival time filter to reduce the number of flights in memory. *...
...//
    public static int arrivalTimeFilter = 21; // Don't consider flights...
... that arrive before 20 UTC
    public static int departTimeFilter = 23;  // Don't consider flights...
... that depart after 24 UTC

    public static int analysisTimeFilterStart = 21;
    public static int analysisTimeFilterEnd = 21;


    /* Seed to use in random number generation */
    public static long randomSeed = 1479;

    /** Use wind in flight time calculations */
    public static boolean useWindInFlightTimeCalculations = false;

    /** Creates a new instance of Settings */
    public Settings() {
    }

    public static String getSettingsFileName(Airspace a){
        // Get the location of the Settings file
        Class cls = a.getClass();
        ClassLoader clsLoader = cls.getClassLoader();
        final String clsAsResource = cls.getName ().replace ('.', File....
...separatorChar).concat (".class");
        java.net.URL location = clsLoader != null ? clsLoader.getResour...
...ce(clsAsResource) :
            ClassLoader.getSystemResource(clsAsResource);
        String locationS = location.toString().trim();
        locationS = locationS.replace("%20"," ");
        locationS = locationS.replace("/", File.separator);
        locationS = locationS.replace("file:","");
        locationS = locationS.replace(clsAsResource, "");

        // Colon (:) is used on Windows platform only
        if(locationS.startsWith(File.separator) && locationS.contains("...
...:")){
            locationS = locationS.substring(1);
        }
        return locationS.concat("AirPlanJ.opts");
    }

    public static void defaultSettings(Airspace a){
//        String settingsFile = Settings.getSettingsFileName(a);
```

```
//
//
//        try{
//            BufferedReader d = new BufferedReader(new FileReader(sett...
...ingsFile));
//            Settings.appDir = d.readLine().trim();
//            Settings.etmsFile = d.readLine().trim();
//            Settings.etmsRadarTrackFile = d.readLine().trim();
//            Settings.etmsFlightPlanFile = d.readLine().trim();
//            Settings.badaDirectory = d.readLine().trim();
//            d.close();
//        }catch(IOException e){
//            System.out.println("Could not read settings file: " + set...
...tingsFile);
//        }

        Settings.recoverProperties();



        /** Operating system. e.g. "Windows XP" */
//        String osName = System.getProperty("os.name");
//        if(osName.equals("Windows XP")){
//            Settings.appDir = "C:\\Documents and Settings\\jeff\\My D...
...ocuments\\Java Projects\\AirPlanJ";
//            Settings.etmsFile = "C:\\Documents and Settings\\jeff\\My...
... Documents\\Drop Box\\Data\\Seagull (sensis) ETMS\\seagull_20040219.tx...
...t";
//            Settings.etmsRadarTrackFile = "C:\\Documents and Settings...
...\\jeff\\My Documents\\Drop Box\\Data\\Flight Explorer Data\\2005\\tz05...
...0829.csv";
//            Settings.etmsFlightPlanFile = "C:\\Documents and Settings...
...\\jeff\\My Documents\\Drop Box\\Data\\Flight Explorer Data\\2005\\fz05...
...0829.csv";
//            Settings.badaDirectory = "C:\\Documents and Settings\\Jef...
...f\\My Documents\\Drop Box\\Analysis\\BADA35\\bada_3_5\\";
//        } else{
//            Settings.appDir = "/Users/antoniotrani/Documents/Jeff/Pro...
...gramming/AirPlanJ";
//            Settings.etmsFile = "/Users/antoniotrani/Documents/Jeff/D...
...ata/Seagull (sensis) ETMS/seagull_20040219.txt";
//            Settings.etmsRadarTrackFile = "/Users/antoniotrani/Docume...
...nts/Jeff/Data/Flight Explorer Data/2005/tz050829.csv";
//            Settings.etmsFlightPlanFile = "/Users/antoniotrani/Docume...
...nts/Jeff/Data/Flight Explorer Data/2005/fz050829.csv";
//            Settings.badaDirectory = "/Users/antoniotrani/Documents/J...
...eff/Analysis/BADA35/bada_3_5/";
```

```
//        }

        //Settings.readRadarTrackDataOnly = true; //false;

        // Time filter - used when writing ExternalTraffic4D for RAMS
        Settings.filterTime = false;

        // Location filter - used when writing data for separation sear...
...ch
        Settings.filterAltitude = true;

        Settings.filterMinTime = 6;
        Settings.filterMaxTime = 24;//24;

        Settings.filterMinAltitude = 180.0;//345.0;
        Settings.filterMaxAltitude = 400.0;//355.0;


        Settings.setDataFiles();

        //Settings.storeProperties();

        Settings.sectorsToExclude = "ZAU,ZOB,ZMA3901,ZMA3402,ZMA3403";

    }

    public static void setDataFiles(){
        Settings.dataDir = appDir + File.separator + "data" + File.sepa...
...rator;
        Settings.errLogFile = dataDir + "error log" + File.separator + ...
..."err.log";
        Settings.sectorFileName = dataDir + "sector" + File.separator +...
... "sectors.csv";
        Settings.sectorMapFileName = dataDir + "sector" + File.separato...
...r + "sectormap.csv";
        Settings.navaidSectorFileName = dataDir + "sector" + File.separ...
...ator + "navaidsector.csv";
        Settings.fixSectorFileName = dataDir + "sector" + File.separato...
...r + "fixsector.csv";
        Settings.airwaySectorIntersectionFileName = dataDir + "sector" ...
...+ File.separator + "airwaySectorIntersection.csv";
        Settings.airportFileName = dataDir + "airports" + File.separato...
...r + "airports.csv";
        Settings.airportTypeFileName = dataDir + "airports" + File.sepa...
...rator + "airportinfo.csv";
        Settings.aircraftTypeFileName = dataDir + "aircraft" + File.sep...
...arator + "acperf.dat";
```

```
        Settings.aircraftPreprocessPerf = dataDir + "aircraft" + File.s...
...eparator + "acperfpreprocess.csv";
        Settings.aircraftClimbDescentFile = dataDir + "aircraft" + File...
....separator + "aircraftClimbDescent.csv";
        Settings.navaidFileName = dataDir + "navaids" + File.separator ...
...+ "navaids.csv";
        Settings.fixFileName = dataDir + "fixes" + File.separator + "fi...
...x.csv";
        Settings.natfixFile = dataDir + "fixes" + File.separator + "nat...
...fix.csv";
        Settings.internationalWaypointsFile = dataDir + "fixes" + File....
...separator + "international_wpts.csv";
        Settings.airwayFileName = dataDir + "airways" + File.separator ...
...+ "airways.csv";
        Settings.internationalAwyFile = dataDir + "airways" + File.sepa...
...rator + "intlairways.csv";
        Settings.ramsFilesDir = dataDir + "RAMS Input Files" + File.sep...
...arator;
        Settings.ramsOutFilesDir = dataDir + "RAMS Output Files" + File...
....separator;
        Settings.outRadarTrackFile = dataDir + "flights" + File.separat...
...or + "flighttracks.cpy";
        Settings.outRadarCoordinates = dataDir + "flights" + File.separ...
...ator + "coordinates.csv";
        Settings.outFlightIndex = dataDir + "flights" + File.separator ...
...+ "flightidx.csv";
        Settings.outFlightIndexCallsign = dataDir + "flights" + File.se...
...parator + "flightcallsignidx.csv";
        Settings.outAltitudeStats = dataDir + "flights" + File.separato...
...r + "altitudestats.csv";
        Settings.outJetRouteFlights = dataDir + "flights" + File.separa...
...tor + "jetrouteflights.csv";
        Settings.windDirectory = dataDir + "wind" + File.separator;
        Settings.windStations = Settings.windDirectory + "station_locat...
...ions.txt";
        Settings.windData = Settings.windDirectory + "srrs-op-050727.tx...
...t";
        Settings.windNavaidClosestStation = Settings.windDirectory + "n...
...avaidStation.csv";
        Settings.windFixClosestStation = Settings.windDirectory + "fixS...
...tation.csv";
        Settings.windNatfixClosestStation = Settings.windDirectory + "n...
...atfixStation.csv";
        Settings.starFile = dataDir + "stars" + File.separator + "STARD...
...P.csv";
        Settings.j6FlightPlanFile = dataDir + "flights" + File.separato...
...r + "fz050829.csv.J6";
```

```
        Settings.ncwfFile = dataDir + "weather" + File.separator + "ncw...
...fp_ncwdp_national_20060611_0500_1hr_cutoff1.gif";
        Settings.kmlFile = dataDir + "kml" + File.separator + "AirPlanJ...
....kml";
        Settings.linkLengthFile = dataDir + "links" + File.separator + ...
..."linkLength.csv";
        Settings.convectionForecastDirectory = dataDir + "weather" + Fi...
...le.separator + "20050727" + File.separator;
        Settings.convectionTopsFilename = Settings.convectionForecastDi...
...rectory + File.separator + "convection_tops.csv";
        Settings.ncwfWeibullParametersFilename = dataDir + "weather" + ...
...File.separator + "ncwf_weibull_parameters.csv";
        Settings.kernelDensityFileName = dataDir + "kernel" + File.sepa...
...rator + "kernel_densities.csv";
        Settings.logFile = dataDir + "log" + File.separator + "AirPlanJ...
....log";
        Settings.apcdmFile = dataDir + "lp" + File.separator + "apcdm.l...
...p";
        Settings.apcdmFlightsInClusterFile = dataDir + "lp" + File.sepa...
...rator + "flightsclusterpath.csv";

        Settings.matlabLibraryFile = appDir + File.separator +  "lib" +...
... File.separator
                    + "matlabdriver" + File.separator + "libmatlab.dll"...
...;

        Settings.matlabJniLibraryFile = appDir + File.separator +  "lib...
..." + File.separator
                    + "matlabdriver" + File.separator + "jnimatlabdrive...
...r.dll";
    }

    public static void storeProperties(){
        APData.properties.setProperty("appDir", Settings.appDir);
        APData.properties.setProperty("etmsFile", Settings.etmsFile);
        APData.properties.setProperty("etmsRadarTrackFile", Settings.et...
...msRadarTrackFile);
        APData.properties.setProperty("etmsFlightPlanFile", Settings.et...
...msFlightPlanFile);
        APData.properties.setProperty("etmsArrivalFile", Settings.etmsA...
...rrivalFile);
        APData.properties.setProperty("etmsDepartureFile", Settings.etm...
...sDepartureFile);
        APData.properties.setProperty("badaDirectory", Settings.badaDir...
...ectory);

//        APData.properties.setProperty("readRadarTrackDataOnly", Strin...
```

```
...g.valueOf(Settings.readRadarTrackDataOnly));
//        APData.properties.setProperty("filterMinAltitude", String.val...
...ueOf(Settings.filterMinAltitude));
//        APData.properties.setProperty("filterMaxAltitude", String.val...
...ueOf(Settings.filterMaxAltitude));
//        APData.properties.setProperty("filterMinTime", String.valueOf...
...(Settings.filterMinTime));
//        APData.properties.setProperty("filterMaxTime", String.valueOf...
...(Settings.filterMaxTime));
//        APData.properties.setProperty("filterTime", String.valueOf(Se...
...ttings.filterTime));
//        APData.properties.setProperty("filterAltitude", String.valueO...
...f(Settings.filterAltitude));

        String settingsFile = Settings.getSettingsFileName(APData.airsp...
...ace);


        try{
            FileOutputStream out = new FileOutputStream(settingsFile);
            APData.properties.store(out, "---AirPlanJ Settings---");
            out.close();

        }catch(IOException e){
            System.out.println("Could not write to file: " + settingsFi...
...le);
        }

    }

    public static void recoverProperties(){
        String settingsFile = Settings.getSettingsFileName(APData.airsp...
...ace);

        try{
            FileInputStream in = new FileInputStream(settingsFile);
            APData.properties.load(in);
            in.close();
        }catch(IOException e){
            System.out.println("Could not load file: " + settingsFile);...
...        }

        Settings.appDir = APData.properties.getProperty("appDir");
        Settings.etmsFile = APData.properties.getProperty("etmsFile");
        Settings.etmsRadarTrackFile = APData.properties.getProperty("et...
...msRadarTrackFile");
        Settings.etmsFlightPlanFile = APData.properties.getProperty("et...
```

```
...msFlightPlanFile");
        Settings.etmsArrivalFile = APData.properties.getProperty("etmsA...
...rrivalFile");
        Settings.etmsDepartureFile = APData.properties.getProperty("etm...
...sDepartureFile");
        Settings.badaDirectory = APData.properties.getProperty("badaDir...
...ectory");

//        Settings.readRadarTrackDataOnly = Boolean.parseBoolean(APData...
....properties.getProperty("readRadarTrackDataOnly"));
//
//        Settings.filterMinAltitude = Double.parseDouble(APData.proper...
...ties.getProperty("filterMinAltitude"));
//        Settings.filterMaxAltitude = Double.parseDouble(APData.proper...
...ties.getProperty("filterMaxAltitude"));
//        Settings.filterMinTime = Integer.parseInt(APData.properties.g...
...etProperty("filterMinTime"));
//        Settings.filterMaxTime = Integer.parseInt(APData.properties.g...
...etProperty("filterMaxTime"));
//
//        Settings.filterTime = Boolean.parseBoolean(APData.properties....
...getProperty("filterTime"));
//        Settings.filterAltitude = Boolean.parseBoolean(APData.propert...
...ies.getProperty("filterAltitude"));

        Settings.setDataFiles();
    }
}
```

TraversalReader.java

```
/*
 * TraversalReader.java
 *
 * Created on September 3, 2007, 12:44 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author jeff
 */
import java.util.*;
import java.io.*;
public class TraversalReader {
    private HashMap<String, FlightTraversal> traversalFlights_;
    private HashMap<String, SectorTraversal> traversalSectors_;

    /** Creates a new instance of TraversalReader */
    public TraversalReader(HashMap<String,FlightTraversal> traversalFli...
...ghts,
            HashMap<String,SectorTraversal> traversalSectors) {
        traversalFlights_ = traversalFlights;
        traversalSectors_ = traversalSectors;
    }

    public static HashMap<String, FlightTraversal> createTraversalFligh...
...ts(){
        return new HashMap<String, FlightTraversal>();
    }

    public static HashMap<String, SectorTraversal> createTraversalSecto...
...rs(){
        return new HashMap<String, SectorTraversal>();
    }

    public TraversalReader readSectorAreaVolume(String fileName){
        try{
            BufferedReader d = new BufferedReader(new FileReader(fileNa...
...me));
            String line;

            while((line = d.readLine()) != null){
                String[] dataFields = line.split(",");
```

```
                if(dataFields.length < 3) continue;

                String sectorId = dataFields[0];
                double sectorArea = Double.parseDouble(dataFields[1]);
                double sectorVolume = Double.parseDouble(dataFields[2])...
...;

                SectorTraversal sectorTraversal = traversalSectors_.con...
...tainsKey(sectorId) ?
                    traversalSectors_.get(sectorId) : new SectorTravers...
...al(sectorId);
                traversalSectors_.put(sectorId, sectorTraversal);

                sectorTraversal.Area = sectorArea;
                sectorTraversal.Volume = sectorVolume;

                sectorTraversal.calculateDensityByMin();
            }

            d.close();

            line = null;

            /** Calculate average traversal time for each sector */
            for(SectorTraversal st:traversalSectors_.values()){
                for(int i=0; i<SectorTraversal.NUM_MINUTES_PER_DAY; i++...
...){
                    st.PlannedDensityByMin2D.set(i,
                            st.PlannedDemandByMin.get(i)/st.Area);
                    st.PlannedDensityByMin3D.set(i,
                            st.PlannedDemandByMin.get(i)/st.Volume);
                }
            }
        }catch(IOException e){
            System.out.println("Error opening file: " + fileName);
        }

        return this;
    }

    public TraversalReader readPlannedSectorCrossing(String fileName){
        try{
            BufferedReader d = new BufferedReader(new FileReader(fileNa...
...me));
            String line;

            while((line = d.readLine()) != null){
```

```
                if(line.startsWith("{")) continue;

                String[] dataFields = line.split(";");

                if(dataFields.length < 12) continue;

                String flightId = dataFields[2].split("_")[0];
                String sectorId = dataFields[1];
                double plannedEntryMin = Double.parseDouble(dataFields[...
...9]);
                double plannedExitMin = Math.min(Double.parseDouble(dat...
...aFields[10]), 1439.0D);
                double plannedTraversalTimeMin = Double.parseDouble(dat...
...aFields[11]);

                FlightTraversal flightTraversal = traversalFlights_.con...
...tainsKey(flightId) ?
                    traversalFlights_.get(flightId) : new FlightTravers...
...al(flightId);
                traversalFlights_.put(flightId, flightTraversal);

                SectorTraversal sectorTraversal = traversalSectors_.con...
...tainsKey(sectorId) ?
                    traversalSectors_.get(sectorId) : new SectorTravers...
...al(sectorId);
                traversalSectors_.put(sectorId, sectorTraversal);

                // Check to see if a flight has been in the sector too ...
...long
                if(plannedExitMin > plannedEntryMin + 30.0D){
                    // Use average sector traversal if it exists
                    double averageSectorTraversal = sectorTraversal.get...
...PlannedTraversalTime();
                    if(averageSectorTraversal > 0.0D){
                        plannedExitMin = plannedEntryMin + averageSecto...
...rTraversal;
                        plannedTraversalTimeMin = averageSectorTraversa...
...l;
                    }else{
                        plannedExitMin = plannedEntryMin + 30.0D;
                        plannedTraversalTimeMin = 30.0D;
                    }
                }

                flightTraversal.PlannedSector.add(sectorId);
                flightTraversal.PlannedEntryMin.add(plannedEntryMin);
                flightTraversal.PlannedExitMin.add(plannedExitMin);
```

```
                flightTraversal.PlannedTraversalTimeMin.add(plannedTrav...
...ersalTimeMin);

                for(long i=Math.round(Math.floor(plannedEntryMin));
                    i <= Math.round(Math.floor(plannedExitMin)); i++){
                    int j = (int)i;
                    double currDemand = sectorTraversal.PlannedDemandBy...
...Min.get(j);
                    currDemand++;
                    sectorTraversal.PlannedDemandByMin.set(j, currDeman...
...d);

                    double currTraversalTime = sectorTraversal.PlannedT...
...otalTraversalTimeByMin.get(j);
                    currTraversalTime += plannedTraversalTimeMin;
                    sectorTraversal.PlannedTotalTraversalTimeByMin.set(...
...j, currTraversalTime);

                    double currNumTraversals = sectorTraversal.PlannedN...
...umTraversalsByMin.get(j);
                    currNumTraversals++;
                    sectorTraversal.PlannedNumTraversalsByMin.set(j, cu...
...rrNumTraversals);
                }
            }

            d.close();

            line = null;

            /** Calculate average traversal time for each sector */
            for(SectorTraversal st:traversalSectors_.values()){
                for(int i=0; i<SectorTraversal.NUM_MINUTES_PER_DAY; i++...
...){
                    st.PlannedAverageTraversalTimeByMin.set(i,
                            st.PlannedTotalTraversalTimeByMin.get(i)/
                            st.PlannedNumTraversalsByMin.get(i));
                }
            }
        }catch(IOException e){
            System.out.println("Error opening file: " + fileName);
        }

        return this;
    }

    public HashMap<String, String> readAirportTypes(String fileName){
```

```
        HashMap<String, String> airportTypes = new HashMap<String, Stri...
...ng>();

        try{
            BufferedReader d = new BufferedReader(new FileReader(fileNa...
...me));
            String line;

            while((line = d.readLine()) != null){
                String[] dataFields = line.split(",");

                String airportId = dataFields[0].trim();
                String airportType = dataFields[2].trim();

                if(airportType.equals("Non-Hub")){
                    airportType = "NonHub";
                }else if(airportType.equals("Large")){
                    airportType = airportId;
                    if(airportType.length() == 4 && (airportType.starts...
...With("K") ||
                            airportType.startsWith("P"))){
                        airportType = airportType.substring(1,4);
                    }
                }

                airportTypes.put(airportId, airportType);
            }

            d.close();
        }catch(IOException e){
            System.out.println("Error opening file: " + fileName);
        }

        return airportTypes;
    }

    public TraversalReader readFlightCluster(String fileName){
        int numFlights = this.traversalFlights_.values().size();
        int flightsRevised = 0;

        try{
            BufferedReader d = new BufferedReader(new FileReader(fileNa...
...me));
            String line;

            while((line = d.readLine()) != null){
                String[] dataFields = line.split(",");
```

```
                if(dataFields.length < 2) continue;

                String flightId = dataFields[0];
                String cluster = "cluster" + dataFields[1];

                if(traversalFlights_.containsKey(flightId)){
                    flightsRevised++;

                    FlightTraversal flight = traversalFlights_.get(flig...
...htId);
                    flight.Airport = cluster;
                }
            }

            d.close();
        }catch(IOException e){
            System.out.println("Error opening file: " + fileName);
        }

        System.out.println(Integer.toString(flightsRevised) + " flights...
... out of "
                + Integer.toString(numFlights) + " revised to use the c...
...luster densities.");

        return this;
    }

    public TraversalReader readAirports(String fileName){
        try{
            BufferedReader d = new BufferedReader(new FileReader(fileNa...
...me));
            String line;

            while((line = d.readLine()) != null){
                String[] dataFields = line.split(" ");

                String flightId = dataFields[1].split("_")[0];
                String airport = dataFields[2];
                if(airport.length() == 4 && (airport.startsWith("K") ||...
... airport.startsWith("P"))){
                    airport = airport.substring(1,4);
                }

                if(traversalFlights_.containsKey(flightId)){
                    FlightTraversal flight = traversalFlights_.get(flig...
...htId);
```

```
                    flight.Airport = airport;
                }
            }

            d.close();
        }catch(IOException e){
            System.out.println("Error opening file: " + fileName);
        }

        return this;
    }

    public TraversalReader readObservedSectorCrossing(String fileName){...
...        try{
            BufferedReader d = new BufferedReader(new FileReader(fileNa...
...me));
            String line;

            while((line = d.readLine()) != null){
                if(line.startsWith("{")) continue;

                String[] dataFields = line.split(";");

                if(dataFields.length < 12) continue;

                String flightId = dataFields[2];
                String sectorId = dataFields[1];
                double observedEntryMin = Double.parseDouble(dataFields...
...[9]);
                double observedExitMin = Math.min(Double.parseDouble(da...
...taFields[10]),1439.0D);
                double observedTraversalTimeMin = Double.parseDouble(da...
...taFields[11]);

                FlightTraversal flightTraversal = traversalFlights_.con...
...tainsKey(flightId) ?
                    traversalFlights_.get(flightId) : new FlightTravers...
...al(flightId);
                traversalFlights_.put(flightId, flightTraversal);

                SectorTraversal sectorTraversal = traversalSectors_.con...
...tainsKey(sectorId) ?
                    traversalSectors_.get(sectorId) : new SectorTravers...
...al(sectorId);
                traversalSectors_.put(sectorId, sectorTraversal);

                // Check to see if a flight has been in the sector too ...
```

```
...long
                if(observedExitMin > observedEntryMin + 30.0D){
                    // Use average sector traversal if it exists
                    double averageSectorTraversal = sectorTraversal.get...
...ObservedTraversalTime();
                    if(averageSectorTraversal > 0.0D){
                        observedExitMin = observedEntryMin = averageSec...
...torTraversal;
                        observedTraversalTimeMin = averageSectorTravers...
...al;
                    }else{
                        observedExitMin = observedEntryMin + 30.0D;
                        observedTraversalTimeMin = 30.0D;
                    }
                }

                flightTraversal.ObservedSector.add(sectorId);
                flightTraversal.ObservedEntryMin.add(observedEntryMin);...
...                flightTraversal.ObservedExitMin.add(observedExitMin);
                flightTraversal.ObservedTraversalTimeMin.add(observedTr...
...aversalTimeMin);

                for(long i=Math.round(Math.floor(observedEntryMin));
                    i <= Math.round(Math.floor(observedExitMin)); i++){...
...                    int j = (int)i;
                    double currDemand = sectorTraversal.ObservedDemandB...
...yMin.get(j);
                    currDemand++;
                    sectorTraversal.ObservedDemandByMin.set(j, currDema...
...nd);

                    double currTraversalTime = sectorTraversal.Observed...
...TotalTraversalTimeByMin.get(j);
                    currTraversalTime += observedTraversalTimeMin;
                    sectorTraversal.ObservedTotalTraversalTimeByMin.set...
...(j, currTraversalTime);

                    double currNumTraversals = sectorTraversal.Observed...
...NumTraversalsByMin.get(j);
                    currNumTraversals++;
                    sectorTraversal.ObservedNumTraversalsByMin.set(j, c...
...urrNumTraversals);
                }
            }

            d.close();
```

```
            line = null;

            /** Calculate average traversal time for each sector */
            for(SectorTraversal st:traversalSectors_.values()){
                for(int i=0; i<SectorTraversal.NUM_MINUTES_PER_DAY; i++...
...){
                    st.ObservedAverageTraversalTimeByMin.set(i,
                            st.ObservedTotalTraversalTimeByMin.get(i)/
                            (st.ObservedNumTraversalsByMin.get(i)+0.000...
...01D));
                }
            }
        }catch(IOException e){
            System.out.println("Error opening file: " + fileName);
        }

        return this;
    }

}
```

TraversalTimeDistribution.java

```
/*
 * TraversalTimeDistribution.java
 *
 * Created on October 22, 2007, 7:40 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author Jeff
 */
import java.util.*;
public class TraversalTimeDistribution {
    private ASector currentSector_;
    private double traversalTime_;
    private KernelDensity kernelDensity_;

    /** Create a new instance of TraversalTimeDistribution */
    public TraversalTimeDistribution(String airportType){
        // Initialize sector data
        currentSector_ = ASector.NullSector;
        traversalTime_ = 0.0D;

        // Start with airport error only
        kernelDensity_ = APData.kernelDensities.get(airportType);
    }

    public KernelDensity run(ASector sector, double traversalTime){
        if(sector.equals(currentSector_)){
            traversalTime_ += traversalTime;
        }else{
            // Get the error distribution for the previous sector
            KernelDensity secDensity = this.getErrorDistributionFromTra...
...versalTime(
                    traversalTime_);

            // Convolute the error
            Convolution convolution = new Convolution(kernelDensity_, s...
...ecDensity);
            kernelDensity_ = convolution.run();

            // Update sector info
            currentSector_ = sector;
            traversalTime_ = traversalTime;
```

```
        }

        return kernelDensity_;
    }

    private KernelDensity getErrorDistributionFromTraversalTime(double ...
...traversalTime){
        if(traversalTime < 4){
            return APData.kernelDensities.get("ratiosLT4");
        }else if(traversalTime < 8){
            return APData.kernelDensities.get("ratiosGT4LT8");
        }else if(traversalTime < 12){
            return APData.kernelDensities.get("ratiosGT8LT12");
        }else if(traversalTime < 16){
            return APData.kernelDensities.get("ratiosGT12LT16");
        }else{
            return APData.kernelDensities.get("ratiosGT16");
        }
    }

}
```

TraversalWriter.java

```
/*
 * TraversalWriter.java
 *
 * Created on September 5, 2007, 11:39 AM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author jeff
 */
import java.util.*;
import java.io.*;
import java.text.*;
public class TraversalWriter {

    /** Creates a new instance of TraversalWriter */
    public TraversalWriter() {
    }

    public TraversalWriter writeTraversalRatioAndDensity(String fileNam...
...e,
            HashMap<String, FlightTraversal> flights, HashMap<String, S...
...ectorTraversal> sectors){

        DecimalFormat format = new DecimalFormat("###0.00000000");

        try{
            PrintWriter out = new PrintWriter(new BufferedWriter(new Fi...
...leWriter(fileName)));

            StringBuilder s = new StringBuilder();

            for(FlightTraversal ft:flights.values()){
                ft.calculateObservedPlannedRatio(sectors);
                for(int i=0; i<ft.ratioValues.size(); i++){
                    ArrayList<Double> ratioData = ft.ratioValues.get(i)...
...;

                    for(int j=0; j<ratioData.size(); j++){
                        s.append(format.format(ratioData.get(j)));
                        if(j == ratioData.size() -1){
                            s.append("\n");
                        }else{
```

```
                        s.append(",");
                    }
                }

            }

        }

        out.print(s.toString());

        out.close();
    }catch(IOException e){
        System.out.println("Could not open file: " + fileName);
    }


    return this;
}

public TraversalWriter writeHitRate(String fileName,
        HashMap<String, FlightTraversal> flights, HashMap<String, S...
...ectorTraversal> sectors){

    DecimalFormat format = new DecimalFormat("###0.0000000");

    try{
        PrintWriter out = new PrintWriter(new BufferedWriter(new Fi...
...leWriter(fileName)));

        StringBuilder s = new StringBuilder();

        for(FlightTraversal flight:flights.values()){
            double hitRate = flight.hitRate();

            if(hitRate < 0.01D) continue;

            s.append(format.format(hitRate));
            s.append(",");
            s.append(format.format(flight.weightedHitRate()));
            s.append(",");
            s.append(format.format(flight.averageDensity(sectors,fa...
...lse)));
            s.append(",");
            s.append(format.format(flight.averageDensity(sectors,tr...
...ue)));
            s.append(",");
            s.append(format.format(flight.totalPlannedTime()));
```

```
                s.append("\n");
            }
            out.print(s.toString());

            out.close();
        }catch(IOException e){
            System.out.println("Could not open file: " + fileName);
        }

        return this;
    }

    public TraversalWriter writeTraversalError(String fileName,
            HashMap<String, FlightTraversal> flights, HashMap<String, S...
...ectorTraversal> sectors){

        DecimalFormat format = new DecimalFormat("###0.00000");

        try{
            PrintWriter out = new PrintWriter(new BufferedWriter(new Fi...
...leWriter(fileName)));

            StringBuilder s = new StringBuilder();

            for(SectorTraversal sector:sectors.values()){
                if(sector.totalObservedDemand() == 0.0D) continue;

                for (int i=0; i<sector.ObservedDemandByMin.size(); i++)...
...{
                    // Exclude the beginning and ending observations
                    if(i < 180 || i > 1260) continue;

                    s.append(sector.SectorId); s.append(",");
                    s.append(format.format(sector.ObservedDemandByMin.g...
...et(i)));
                    s.append(",");
                    s.append(format.format(sector.PlannedDemandByMin.ge...
...t(i)));
                    s.append(",");
                    s.append(format.format(sector.DeterministicDemand10...
...minLAT.get(i)));
                    s.append(",");
                    s.append(format.format(sector.DeterministicDemand30...
...minLAT.get(i)));
                    s.append(",");
                    s.append(format.format(sector.DeterministicDemand60...
...minLAT.get(i)));
```

```
                    s.append(",");
                    s.append(format.format(sector.ProbabilisticDemand10...
...minLAT.get(i)));
                    s.append(",");
                    s.append(format.format(sector.ProbabilisticDemand30...
...minLAT.get(i)));
                    s.append(",");
                    s.append(format.format(sector.ProbabilisticDemand60...
...minLAT.get(i)));
                    s.append("\n");
                }
            }
            out.print(s.toString());

            out.close();
        }catch(IOException e){
            System.out.println("Could not open file: " + fileName);
        }

        return this;
    }

}
```

UpdateTimeAndAltitude.java

```
/*
 * UpdateTimeAnAltitude.java
 *
 * Created on October 22, 2007, 1:29 AM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author Jeff
 */
import java.util.*;
public class UpdateTimeAndAltitude {
    private double currentTime_;
    private double currentDistance_;
    private double currentAltitude_;
    private Link3D outLink_;
    private EAircraft ac_;
    private double cruiseFlightLevel_;
    private double previousTime_;
    private Scenario.FlightStatus flightStatus_;
    private Node3D<Double> destinationAirportNode_;

    /** Creates a new instance of UpdateTimeAnAltitude */
    public UpdateTimeAndAltitude(double currentTime, double currentDist...
...ance,
            double currentAltitude, Link3D outLink, EAircraft ac,
            double cruiseFlightLevel, Scenario.FlightStatus flightStatu...
...s,
            Node3D<Double> destinationAirportNode) {
        currentTime_ = currentTime;
        currentDistance_ = currentDistance;
        currentAltitude_ = currentAltitude;
        outLink_ = outLink;
        ac_ = ac;
        cruiseFlightLevel_ = cruiseFlightLevel;
        flightStatus_ = flightStatus;
        destinationAirportNode_ = destinationAirportNode;
    }

    public double getCurrentTime(){
        return currentTime_;
    }
```

```
    public double getCurrentDistance(){
        return currentDistance_;
    }

    public double getCurrentAltitude(){
        return currentAltitude_;
    }

    public double getPreviousTime(){
        return previousTime_;
    }

    public Scenario.FlightStatus getFlightStatus(){
        return flightStatus_;
    }

    public UpdateTimeAndAltitude run(){
        // Get direction of link
        double linkAzimuth = outLink_.getAzimuth();

        // Wind variables
        double windSpeed = outLink_.getWindSpeed(currentTime_, currentA...
...ltitude_);
        double windDirection = outLink_.getWindDirection(currentTime_, ...
...currentAltitude_);

        // First find the angle (theta) between the wind and aircraft a...
...zimuth using
        // the dot product
        double angle_a = Math.toRadians(linkAzimuth);
        double angle_b = Math.toRadians(windDirection);
        double dot_a_b = Math.sin(angle_a) * Math.sin(angle_b) + Math.c...
...os(angle_a) * Math.cos(angle_b);
        double theta = Math.acos(dot_a_b);

        // Convert wind speed from nm/hr to nm/min
        windSpeed = windSpeed / 60.0D;

        // Use the cosine law to calculate groundspeed
        double i_actr = 1 / ac_.getAverageClimbTimeRoc(cruiseFlightLeve...
...l_);
        double i_actr2 = Math.sqrt(i_actr*i_actr + windSpeed*windSpeed ...
...-
                2*i_actr*windSpeed*Math.cos(theta));
        double averageClimbTimeRoc2 = 1 / i_actr2;

        double i_campnm = 1 / ac_.getCruiseAirspeedMinPerNm(cruiseFligh...
```

```
...tLevel_);
        double i_campnm2 = Math.sqrt(i_campnm*i_campnm + windSpeed*wind...
...Speed -
                2*i_campnm*windSpeed*Math.cos(theta));
        double cruiseAirspeedMinPerNm2 = 1 / i_campnm2;

        double i_adtr = 1 / ac_.getAverageDescentTimeRoc(cruiseFlightLe...
...vel_);
        double i_adtr2 = Math.sqrt(i_adtr*i_adtr + windSpeed*windSpeed ...
...-
                2*i_adtr*windSpeed*Math.cos(theta));
        double averageDescentTimeRoc2 = 1 / i_adtr2;

        // Other aircraft performance interpolations
        double averageClimbAltRoc = ac_.getAverageClimbAltRoc(cruiseFli...
...ghtLevel_);
        double averageDescentAltRoc = ac_.getAverageDescentAltRoc(cruis...
...eFlightLevel_);

        // Length (distance) of link in nm
        double linkLength = outLink_.getLength();

        // Update the current distance from the origin airport
        currentDistance_ += linkLength;

        // Interpolated distances
        double climbDist = ac_.getClimbDistance(cruiseFlightLevel_);
        double descentDist = ac_.getDescentDistance(cruiseFlightLevel_)...
...;
        // Update time and altitude
        previousTime_ = currentTime_;


        if(flightStatus_.equals(Scenario.FlightStatus.CLIMBING) &&
                currentDistance_ < climbDist){
            currentTime_ += averageClimbTimeRoc2 * linkLength;
            currentAltitude_ += averageClimbAltRoc * linkLength;
        }else if(flightStatus_.equals(Scenario.FlightStatus.CLIMBING)){...
...             // Cruising distance for link
            double linkCruiseDist = currentDistance_ - climbDist;

            // Climbing distance for link
            double linkClimbDist = linkLength - linkCruiseDist;

            // Update the flight time
            currentTime_ += averageClimbTimeRoc2 * linkClimbDist +
                    cruiseAirspeedMinPerNm2 * linkCruiseDist;
```

```
            // Flight will reach cruising altitude on the link
            currentAltitude_ = cruiseFlightLevel_;

            // Update flags for climbing and cruising
            flightStatus_ = Scenario.FlightStatus.CRUISING;
        }else if(flightStatus_.equals(Scenario.FlightStatus.CRUISING)){...
...             Node3D<Double> toNodeTmp = outLink_.getToNode();

            Mapping m = new Mapping(toNodeTmp.getLatitude(), toNodeTmp....
...getLongitude(),
                    destinationAirportNode_.getLatitude(),
                    destinationAirportNode_.getLongitude());
            //m.calculateDistance();

            double distToAptNm = m.getDistance();

            if(distToAptNm > descentDist){
                // Cruising during entire link
                currentTime_ += cruiseAirspeedMinPerNm2 * linkLength;
            }else{
                // Descending during a portion of the link

                // Descending distance for link
                double linkDescentDist = descentDist - distToAptNm;

                // Cruising distance for link;
                double linkCruiseDist = linkLength - linkDescentDist;

                // Update flight time
                currentTime_ += cruiseAirspeedMinPerNm2 * linkCruiseDis...
...t +
                        averageDescentTimeRoc2 * linkDescentDist;

                // Update altitude
                currentAltitude_ += averageDescentAltRoc * linkDescentD...
...ist;

                // Update flags for cruising and descending
                flightStatus_ = Scenario.FlightStatus.DESCENDING;
            }

        }else if(flightStatus_.equals(Scenario.FlightStatus.DESCENDING)...
...){
            currentTime_ += averageDescentTimeRoc2 * linkLength;
            currentAltitude_ += averageDescentAltRoc * linkLength;
        }
```

```
        return this;
    }

}
```

Wind.java

```
/*
 * Wind.java
 *
 * Created on May 26, 2007, 4:54 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
//package AirPlanJ;
/**
 *
 * @author Jeff
 */
import java.util.*;
public class Wind {
    public static final int[] startTime = {0,600,1200,1800};
    public static final int[] endTime = {600,1200,1800,0};
    public static final int[] flightLevel = {30,60,90,120,180,240,300,3...
...40,390};

    /* Number of wind stations (fixed) */
    private static final int NUM_WIND_STATIONS = 227;

    private static final float INIT_HASH_LOAD_FACTOR = 0.25F;

    /** Data structure to hold wind data for one day */
    private HashMap<String, WindData> windData_;

    /** Closest wind station to each  */
    private HashMap<String, String> closestStationToIdentifier_;


    /** Creates a new instance of Wind */
    public Wind() {
        windData_ = new HashMap<String, WindData>(NUM_WIND_STATIONS,
                Settings.hashMapLoadFactor);
        closestStationToIdentifier_ = new HashMap<String, String>(
                Settings.hashMapDefaultInitialCapacity,
                Settings.hashMapLoadFactor);
    }

    class WindData{
        public int[][] windSpeed;
        public int[][] windDirection;
        public String[][] windString;
```

```
        public double latitude;
        public double longitude;

        public WindData(){
            this.windSpeed = new int[4][9];
            this.windDirection = new int[4][9];
            this.windString = new String[4][9];

        }
    }

    /* Store location of each wind station */
    public Wind addWindStation(String station, double latitude, double ...
...longitude){
        WindData wd = new WindData();
        wd.latitude = latitude;
        wd.longitude = longitude;

        windData_.put(station, wd);

        return this;
    }

    /* Storing wind data */
    public Wind addWindData(String station, String windString, int star...
...tTime, int flightLevel){
        int timeIdx = 0;
        int flIdx = 0;
        switch (startTime){
            case    0: timeIdx = 0; break;
            case  600: timeIdx = 1; break;
            case 1200: timeIdx = 2; break;
            case 1800: timeIdx = 3; break;
            default  : timeIdx = -1; break;
        }

        switch (flightLevel){
            case  30: flIdx = 0; break;
            case  60: flIdx = 1; break;
            case  90: flIdx = 2; break;
            case 120: flIdx = 3; break;
            case 180: flIdx = 4; break;
            case 240: flIdx = 5; break;
            case 300: flIdx = 6; break;
            case 340: flIdx = 7; break;
            case 390: flIdx = 8; break;
            default : flIdx = -1; break;
```

```
        }

        int windSpeed_ = 0;
        int windDirection_ = 0;

        // Check for whitespace
        windString = windString.trim();
        if(windString.length() > 0){
            // Only use first four characters for wind direction and sp...
...eed
            windDirection_ = Integer.parseInt(windString.substring(0,2)...
...);
            windSpeed_ = Integer.parseInt(windString.substring(2,4));

            // Winds over 100 knots
            if(windDirection_ > 36 && windDirection_ < 99){
                windDirection_ -= 50;
                windSpeed_ += 100;
            } else if(windDirection_ == 99){
                windDirection_ = 0;
            }

            // Convert to degrees
            windDirection_ *= 10;
        }

        //
        WindData wd = windData_.get(station);
        wd.windDirection[timeIdx][flIdx] = windDirection_;
        wd.windSpeed[timeIdx][flIdx] = windSpeed_;
        wd.windString[timeIdx][flIdx] = windString;

        return this;
    }

    /* Get wind stations */
    public Iterator<String> getWindStationIterator(){
        Set<String> s = windData_.keySet();
        return s.iterator();
    }
    /** Return the wind data struct by station id */
    public WindData getWindData(String station){
        return windData_.get(station);
    }

    /** Check to see if station identifier is valid */
    public boolean containsWindStation(String station){
```

```
        return windData_.containsKey(station);
    }

    public double getWindStationLatitude(String station){
        return windData_.get(station).latitude;
    }

    public double getWindStationLongitude(String station){
        return windData_.get(station).longitude;
    }

    /** Get wind direction */
    public int getWindDirection(String station, int startTimeIdx, int f...
...lightLevelIdx){
        return windData_.get(station).windDirection[startTimeIdx][fligh...
...tLevelIdx];
    }

    /** Get wind speed */
    public int getWindSpeed(String station, int startTimeIdx, int fligh...
...tLevelIdx){
        return windData_.get(station).windSpeed[startTimeIdx][flightLev...
...elIdx];
    }

    public String getWindString(String station, int startTimeIdx, int f...
...lightLevelIdx){
        return windData_.get(station).windString[startTimeIdx][flightLe...
...velIdx];
    }

    public Wind addClosestStationToIdentifier(String identifier, String...
... station){
        closestStationToIdentifier_.put(identifier, station);

        return this;
    }

    /** Get wind direction using airspace identifier */
    public int getWindDirectionByAirspaceIdentifier(String identifier, ...
...double timeMinutes,
            double flightLevelD){

        // Return a default value of 0 if the airspace identifier is no...
...t
        // recognized.
        if(!closestStationToIdentifier_.containsKey(identifier)) return...
```

```
... 0;

        String stationId = closestStationToIdentifier_.get(identifier);...
...        int timeIndex = getTimeIndex(timeMinutes);
        int flightLevelIndex = getFlightLevelIndex(flightLevelD);

        return windData_.get(stationId).windDirection[timeIndex][flight...
...LevelIndex];
    }

    private int getTimeIndex(double timeMinutes){
        int timeIndex = 0;
        for(int i=0; i<startTime.length; i++){
            if(timeMinutes >= startTime[i] && timeMinutes <= endTime[i]...
...)
                timeIndex = i;
        }

        return timeIndex;
    }

    private int getFlightLevelIndex(double flightLevelD){
        int flightLevelIndex = 0;
        for(int i=0; i<flightLevel.length; i++){
            if(flightLevelD > flightLevel[i]) flightLevelIndex = i;
        }

        return flightLevelIndex;
    }

    /** Get wind speed using airspace identifier */
    public int getWindSpeedByAirspaceIdentifier(String identifier, doub...
...le timeMinutes,
            double flightLevelD){

        // Return a default value of 0 if the airspace identifier is no...
...t
        // recognized.
        if(!closestStationToIdentifier_.containsKey(identifier)) return...
... 0;

        String stationId = closestStationToIdentifier_.get(identifier);...
...        int timeIndex = getTimeIndex(timeMinutes);
        int flightLevelIndex = getFlightLevelIndex(flightLevelD);

        return windData_.get(stationId).windSpeed[timeIndex][flightLeve...
...lIndex];
```

```
    }

}
```

APPENDIX F: AirPlanJ MANUAL

The Java program AirPlanJ is non-interactive and requires only the Java heap size and the AirPlanJ.opts options file to be set. The heap size is set with the -Xms option for the initial heap size and the -Xmx option for the maximum heap size. The following invokes AirPlanJ with a 1500 MB heap size.

```
java -Xms1500m -Xmx1500m AirPlanJ
```

The options file (AirPlanJ.opts) must be in the same directory as the Java class (AirPlanJ.class) or jar (AirPlanJ.jar) file. The standard options are shown below.

Data Field	Description
appDir	Root directory of the application
badaDirectory	Directory of the BADA files
etmsFlightPlanFile	Fully qualified location of the ETMS flight plan (FZ) file
etmsRadarTrackFile	Fully qualified location of the ETMS radar track (TZ) file
etmsArrivalFile	Fully qualified location of the ETMS flight arrival (AZ) file
etmsDepartureFile	Fully qualified location of the ETMS flight departure (DZ) file

An example options file is included below.

```
#---AirPlanJ Settings---
#Sat Aug 04 02:31:06 EDT 2007
appDir=C\:\\Documents and Settings\\Jeff\\My Documents\\Drop
    Box\\Programming\\AirPlanJ
badaDirectory=C\:\\Documents and Settings\\Jeff\\My Documents\\Drop
    Box\\Analysis\\BADA35\\bada_3_5\\
etmsFlightPlanFile=C\:\\Documents and Settings\\Jeff\\My Documents\\Drop
    Box\\Data\\Flight Explorer Data\\2005\\fz050727.csv
etmsRadarTrackFile=C\:\\Documents and Settings\\Jeff\\My Documents\\Drop
    Box\\Data\\Flight Explorer Data\\2005\\tz050727.csv
etmsArrivalFile=C\:\\Documents and Settings\\Jeff\\My Documents\\Drop
    Box\\Data\\Flight Explorer Data\\2005\\az050727.csv
etmsDepartureFile=C\:\\Documents and Settings\\Jeff\\My Documents\\Drop
    Box\\Data\\Flight Explorer Data\\2005\\dz050727.csv
```