Modifying TRANSIMS (Transportation Analysis and Simulation) to Include Dynamic Value Pricing and Departure Time Choice

Kwang-Sub Lee

Dissertation submitted to the Faculty of
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirement for the degree of

DOCTOR OF PHILOSOPY
In
Civil and Environmental Engineering

Dr. Antoine G. Hobeika, Chair
Dr. Hesham A. Rakha
Dr. Antonio A. Trani
Dr. Hojong Baik
Dr. Barbara M.P. Fraticelli

June 10, 2009
Blacksburg, Virginia

Keywords: HOT lane value pricing, dynamic toll pricing, heterogeneous value of time, traveler response to value pricing, route choice, departure time choice, TRANSIMS

Copyright 2009, Kwang-Sub Lee

Modifying TRANSIMS (Transportation Analysis and Simulation) to Include Dynamic Value Pricing and Departure Time Choice

Kwang-Sub Lee

ABSTRACT

Value pricing is now an accepted strategy for congestion and demand management in metropolitan areas. Along with alternate congestion management strategies, many transportation agencies have started looking at value pricing as a method to help financial shortfalls of new congestion management projects. Value pricing allows revenue collected from toll facilities to reduce operational concerns with underutilized High Occupancy Vehicle (HOV) facilities and relieves environmental concerns by reducing travel demand. Recently, transportation agencies have become increasingly interested in a high-occupancy toll (HOT) lane value pricing system with time-dependent tolls or dynamic tolls that change by the congestion level. However, there is a lack of proper travel demand forecasting tools that can evaluate and determine the impacts of pricing on travelers' decision in relation to congestion. The current methods use aggregated and zonal based approaches that lack the capability of tracing individual travelers through the supply network in order to capture his/her travel decisions as it pertains to the estimated cost for toll usage. The conventional models do not consider individual traveler socio-economic characteristics, particularly the heterogeneous value of time (VOT).

TRANSIMS (Transportation Analysis Simulation System) differs from current travel demand forecasting methods in its underlying concepts and structure. These differences include a consistent and continuous representation of time, a detailed representation of persons and households, time-dependent routing, and a person-based Microsimulator. The TRANSIMS Microsimulator is the only simulation tool that maintains the identity of the traveler throughout the simulation and is capable of accessing the database of each individual (e.g., income, age, trip purpose). It traces the movement of people as well as vehicles on a second-by-second basis. Although TRANSIMS environment has significantly improved over the past few years, there are still issues that need to be improved upon including: the pricing of a HOT lane with dynamic tolls and the rescheduling of activities (i.e., departure time choice model) in response to network conditions.

The primary objectives of this study are to improve functions of TRANSIMS by modifying source codes in order to utilize non-linear, individual VOT function in route choice of a HOT lane value pricing system, to implement 15-min dynamic tolls that vary by level of service (i.e., volume/capacity ratio) in the HOT lane(s) and to develop departure time choice model. Testing the proposed methodologies using real-world data as case studies and evaluating the impacts of dynamic tolls and/or departure time choice model are other objectives of this study. The test site of the HOT lane system is a segment of I-5 northbound from Hwy 217 to I-405 near the central business district (CBD) in Portland metropolitan region, Oregon.

The experimental analyses of the application of dynamic tolls and individual VOT demonstrate the feasibility of the proposed simulation methodology. The outputs from the microscopic analysis clearly indicate the effectiveness of the analysis in scrutinizing travelers' route choice behavior based on different socio-economic and travel characteristics when different toll rates are applied. The effects of individual VOT on route choice are consistent with intuition; that is, travelers with higher VOTs are more likely to choose the HOT lane(s). In addition, the impacts of various tolls on route choice are analyzed on the basis of socio-economic and trip characteristics of each traveler.

In addition to the development of the dynamic value pricing along with individual VOT, the departure time choice model is also developed. The proposed method is a post-processing of route choice and represents a sequential decision making process of travelers who want to depart early or late based on congestion, individual attributes and activity characteristics. This paper presents the results of a departure time choice model and its impacts on a HOT lane system using Portland, Oregon as a case study. The results show that 13.9% of households did change their departure time because of congestion and/or tolls.

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere gratitude and respect to my supervisor, Dr. Antoine G. Hobeika, for his invaluable academic guidance and support during my Ph.D. Under his supervision for seven years, I have learned a lot of different disciplines from him. I also would like to thank my committee members, Dr. Hesham Rakha, Dr. Antonio A. Trani, Dr. Barbara M.P. Fraticelli and Dr. Hojong Baik for their guidance, discussion and enthusiasm on my research.

I would like to express special thanks to Dr. Hojong Baik. From the very beginning of my Ph.D. to the present, he is mentally with me all the time whenever I have difficult times or need an advice. He is very generous and patient, and gave me very helpful advice.

I can not thank enough my parents, Dong-il Lee and Jung-ae Kim. Without their never-ending love, supports and prayers, I would not have come to the United States and would not have finished my study.

I also would like to thank my beloved wife, Jiyoung Kim, and a beautiful daughter, Chloe Lee, for their support and patience. I appreciate your understanding, love and prayer on me. I would like to express thanks to God to meet and marry my wife and have a precious daughter.

I would like to thank other supporters, including Dong-Hyuk Sohn for his discussion with me, Danielle Jolly for her proofreading, Yoon-Soo Lee and Hyunseung Yoo for their assistance on C++ programming, and Korean members in Civil and Environmental Engineering at Virginia Tech.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1. Introduction

1.1 Background

Value pricing, also known as congestion pricing, is a demand-based congestion management strategy using market principles. It burdens road users in peak periods with higher tolls, which may vary with the level of congestion. Value pricing strategy provides drivers on the road during peak travel periods with incentives to shift their trips to off-peak travel times, less congested routes, alternative modes, or shared driving. It can also solve the under-utilization of High Occupancy Vehicle (HOV) lanes. For example, the High Occupancy Toll (HOT) lanes system can improve economic efficiency by utilizing wasted space on the under-utilized lanes. In addition, imposing value pricing provides new revenue for highways, public transit investment opportunities, as well as reduces environmental emissions.

The technical advancements in electronic toll collection (ETC) systems make the implementation of value pricing systems feasible and attractive. Drivers do not need to stop at toll plazas in recent pricing systems because fees are electronically collected via Automated Vehicle Identification (AVI) transponders and overhead readers. With the system, drivers simply put small tags or transponders in the windshield of their cars. Tolls are then collected as the tag is read by electronic scanners suspended from overhead gantries as the vehicle passes by at normal highway speeds. Travelers are required to obtain and manage their accounts using a credit card or through a quick phone call, visit to a kiosk or office, or use of a website. Tags emit a signal warning consumers when their account is running low, or they are informed through messages beamed to them as they go by a toll collection point.

Toll roads that implement value pricing (i.e., HOT lanes) are different from traditional toll roads. On traditional toll roads, drivers have to pay tolls for use of roads without options on selecting a toll lane or a non-toll lane. On the other hand, drivers have some degree of choice over whether they travel on the toll lane by paying tolls or travel on the non-toll lane without charges in a HOT lane system. Value pricing projects are also different from classic toll roads in terms of varying tolls depending on time of day or congestion level, whereas traditional toll roads typically have flat tolls twenty-four hours a day, seven days a week.

A number of value pricing projects have been introduced to relieve peak-period congestion in the U.S. especially since the 1990s. One of the existing value pricing systems is State Route 91 Express Lane in Orange County, CA, which is the first application of value pricing in the U.S. Another example is the HOT lanes of Interstate 15 (FasTrak Program), which utilizes dynamic toll rates depending on level of service (LOS) in HOT lanes.

1.2 Problem Statement

Value pricing has been extensively studied by both transportation researchers and economists under different objectives, assumptions, perspectives, and/or modeling approaches. However, the existing analytical or simulation-based analyses and models have their own strengths and weaknesses. For example, value pricing analyses based on economic theory of marginal-cost principle are theoretically sound, but are limited to hypothetical and idealized conditions, and the assumptions made are not practical. These approaches do not take into account the effects of time and space upon traffic dynamics and do not incorporate them into the transportation planning process. The analytical models based on the static traffic assignment techniques using link performance functions cannot accurately compute the link travel time due to their inability to capture the realistic dynamics of traffic flows. Thus, they are not appropriate to model and evaluate value pricing especially with dynamically varying toll rates. In addition, the analytical models are constrained by the developed solution algorithms, which make them inappropriate to apply to real-size transportation networks due to the complexities (e.g., non-linearity and non-convexity) of the problem formulations and solution procedures. A few problems identified in the analytical models could be solved by using a simulation-based approach. However, most simulation-based value pricing models failed to account for heterogeneity of travelers in terms of value of time (VOT), because they use aggregate data and do not have the ability to consider individual socio-economic information.

While pricing options are considered, there is a lack of proper travel demand forecasting (TDF) tools that can efficiently evaluate and determine the impacts of pricing on travel behavior and consequently congestion. The current TDF models are not capable of addressing issues that relate to individual traveler behavior choices when faced with decisions regarding various pricing options such as tolls that vary dynamically. The conventional models are aggregate and zonal based approaches that lack the capability of tracing individual travelers through the supply network in order to capture his/her decisions regarding certain pricing options.

One of the critical factors in modeling value pricing is the determination of VOT. Many researchers have reached the conclusion that individuals react differently to pricing strategies depending on their VOTs that are related to their socio–economic and trip characteristics (Ghosh, 2000; Steimetz and Brownstone, 2005). De Palma and Lindsey (2004) emphasized that "*individuals differ in their VOT and preferences for where and when to travel, as well as with respect to socio-economic characteristic that may be relevant for social policy in general.*" However, most available studies assume a single class of users (i.e., constant VOT over the population), or at best two or three income classes of users (Hensher and Goodwin, 2004; Small, 1992a; Ghosh, 2000). While certain studies have used heterogeneous VOT, they make a simple assumption of VOT as a random variable distributed across the population of travelers, based on a certain probability density function or log-normal distribution (Armstrong, et al., 2001; Ortuzar and Willumsen, 2001; Lam, 2004; Mahmassani et al., 2005). Theses approaches cannot fully capture each traveler's VOT and relate it to his/her socio-economic and/or travel characteristics.

TRANSIMS (Transportation Analysis Simulation System) is an activity-based and an integrated system of travel forecasting and microscopic simulation models developed for regional transportation planning. The TRANSIMS environment has significantly changed over the past few years. TRANSIMS is now open to the public. The software, source codes, tutorials, documents, and case studies are available throughout the TRANSIMS Open Source website (2008). New TRANSIMS software (since version 4.0) is now available for both Windows and Linux. The Federal Highway Administration

(FHWA) and open source community members continue working to improve TRANSIMS codes, methods and documentation. However, there are still a few issues that need to be improved. For example, even though new TRANSIMS allows tolls (i.e., constant or time-varying tolls), the existing Router accepts only a constant factor to convert cost value to travel time value, thus does not allow various VOTs. There is currently no activity rescheduling process, including departure time choice, in response to network conditions.

The activity rescheduling process in response to congestion/toll based on updated link travel times is important for transportation planning studies particularly in metropolitan area where significant congestion occurs in the peak period. Some travelers would adjust their departure times to avoid delay or a toll road in the value pricing system. Over the past decades the departure time choice models are well studied due to the importance of dealing with peak-period traffic congestion. The dominant model of departure time choice in practice and academic research is a discrete choice model. Other types of models include continuous models, joint models of departure time choice combined with mode and/or route choice, departure time choice decisions under congestion pricing, and activity rescheduling models. However, TRANSIMS has not yet implemented the departure time choice model. The old version of TRANSIMS (i.e., v.3.0.2) had a feedback module of the Activity Regenerator. The two primary purposes of this module include: 1) Activity Generator-to-Activity Generator feedback to clean up and reassign activities that cannot be assigned or completed by the Activity Generator, and 2) Router-to-Activity Generator feedback to solve a problem associated with activities that the Router cannot assign paths. However, there was no module for an activity rescheduling process in response to network conditions. Although functions of TRANSIMS has been improved, the existing TRANSIMS (v.4.0.1) still does not have a module for activity rescheduling, including departure time choice model, in response to network conditions. For better understanding of the travelers' behaviors in response to congestion/toll, it is important to improve TRANSIMS functions by including the departure time choice model for the application of the HOT lane value pricing.

1.3 Research Objectives

The primary objectives of this study are: 1) to utilize non-linear, individual VOT function in route choice of a HOT lane value pricing system; 2) to implement 15-min dynamic tolls that vary by level of service (i.e., volume/capacity ratio) in the HOT lane(s); and 3) to develop departure time choice model to improve TRANSIMS functions. To achieve these objectives, TRANSIMS simulation model was utilized and source codes were added and/or modified. TRANSIMS can analyze dynamic HOT lane value pricing in a large transportation network by taking into account an individual-based VOT function. TRANSIMS differs from current travel demand forecasting methods in its underlying concepts and structure. These differences include a consistent and continuous representation of time; a detailed representation of persons and households; time-dependent routing; and a person-based Microsimulator. TRANSIMS Microsimulator is the only simulation tool that maintains the identity of the traveler throughout the simulation, and is capable of accessing the database of each individual (e.g., income, age, trip purpose, etc.). In other words, it traces the movement of people as well as vehicles on a second-by-second basis. In addition, TRANSIMS Route Planner utilizes a time-dependent, individually-based route choice model that is suitable for considering each individual VOT. TRANSIMS is also an activity-based simulation model that has the advantage of analyzing the effect of value pricing on each traveler activity patterns.

The main contributions of this research can be summarized as follows: 1) the enhancement of the Router and associated modules of TRANSIMS to accept dynamic value pricing; 2) the improvement of the Router to accept individual VOT; 3) the development of departure time choice model in response to congestion/toll; 4) the evaluation of the effects of dynamic value pricing and the analysis of travelers' behavior on route choice; 5) the evaluation of the travelers behavior in terms of departure time adjustment; and 6) the practical applications of the proposed methods using real-world data from Portland, OR as case studies. The test site of the HOT lane system is a segment of I-5 northbound from Hwy 217 to I-405 near the central business district (CBD) in Portland metropolitan region, Oregon.

1.4 Organization of Dissertation

This dissertation consists of six chapters. Following the introduction, the second chapter presents background, concepts and types of value pricing and summarizes some of the existing value pricing systems in the U.S. The third chapter reviews the existing analytical and simulation approaches to solve value pricing model, and presents literature review on VOT and departure time choice models. The fourth chapter presents a non-linear, individual VOT function and the generalized travel time function that simultaneously considers each individual VOT and 15-min dynamic tolls. The proposed simulation methodology and specific modifications for enhancements to TRANSIMS are also discussed, following by the two experimental analyses as case studies. The fifth chapter presents the proposed methodology for the development of the departure time choice model. The simulation data and alternative scenarios are discussed, followed by simulation results. The summary and conclusions are presented in the last chapter.

Chapter 2. What is Value Pricing

2.1 Background

A number of value pricing projects have been introduced to relieve peak-period congestion especially since the 1990s in the United States. The Federal Highway Administration's (FHWA) Congestion Pricing Pilot Program was the first attempt to introduce variable pricing to highways and bridges. One of the projects funded by FHWA was the San Francisco-Oakland Bay Bridge project. Following the Program, the Value-Pricing Pilot Program was initiated by Congress under the Transportation Equity Act for the 21st Century (TEA-21). This is an ongoing effort, following the Congestion-Pricing Pilot Program under the Intermodal Surface Transportation Efficiency Act of 1991 (ISTEA). The purpose of Value-Pricing Pilot Program is to demonstrate and evaluate road and parking pricing concepts that achieve significant and lasting reductions in highway congestion (Value Pricing Pilot Program, 2009). Under the program, public agencies continuously apply for federal grants for the purpose of investigating innovative pricing approaches for reducing congestion. In terms of published literature, Humphrey Institute for Public Affairs at the University of Minnesota introduced value pricing to the general public through several publications. They conducted focus groups and surveys on value pricing concepts. The National Research Council (1994) also examined value pricing, and published a special report.

2.2 Types of Road Pricing and Pricing Methods

2.2.1 Types of pricing

There are three types of pricing: road pricing, vehicle use pricing, and parking pricing. However, the focus of this dissertation is on road pricing. Road pricing includes congestion pricing (variable tolls on toll facilities), value pricing including HOT lanes, Fast and Intertwined Regular (FAIR) lanes, and cordon tolls (HHH, 2009).

2.2.1.1 Congestion pricing (variable tolls on toll facilities)

Congestion pricing is contingent upon peak travel times which cause variability in toll pricing. For instance, tolls are significantly higher during congested periods, while they are lower during un-congested periods. It is, thus, called “time variable pricing.” The critical difference from value pricing is that drivers can choose when to drive through the toll plaza; however, drivers do not have an option to choose between the toll lane and the regular free lane (Smith, 2002). Therefore, travelers passing through a toll facility must pay the required toll regardless of their lane unless they are in a designated free HOV lane. The major purpose of using variable tolls on toll facilities is to encourage travelers to shift to other times, routes, and modes during peak travel times; thereby, facilitating traffic control. This type of value pricing has great potential opportunities, because many existing bridges and tunnels in the U.S. have a toll infrastructure in place.

2.2.1.2 Value pricing

The major difference in value pricing from congestion pricing is that value pricing offers drivers the option of driving a toll road or not. There are two types of value pricing experiments, including HOT lanes and FAIR lanes.

- **HOT lanes**

HOT lane is a term used to designate certain special use lanes on an otherwise free highway facility (HHH, 2009). The basic concept of HOT lane is to provide low occupancy vehicles permission to use the fast lane(s) by paying a toll, while HOVs use the fast lanes for free or at a discounted toll rate. HOT lane system is one of the most popular types of value pricing because it provides several advantages in terms of cost effectiveness, ease of operation, and the justification to not to construct new roads. The HOT lane concept can be implemented basically in two ways. One is to convert current HOV lanes to HOT lanes. The existing examples implemented in the U.S. are Interstate 15 in San Diego, California and Interstate 10 in Houston, Texas. The former was introduced to solve the problem of underutilization of HOV lanes, by allowing single occupant vehicles (SOVs) paying a toll to use the HOT lanes. The other way of implementing HOT lanes is to construct new lanes on an existing highway facility as implemented on State Route 91 in California.

- **FAIR (Fast and Intertwined Regular) lanes**

FAIR lanes are considered an innovative form of value pricing. This concept stems from political and public objections about value pricing, which converts existing free lanes on a freeway to value priced lanes (HHH, 2009). The most significant advantage of FAIR lanes is that all road users can directly have benefits with increased choices. Travelers who choose to pay for the use of a tolled lane benefit from reduced congestion. On the other hand, those who use general-purpose lanes face more congestion, but they are compensated with credits for giving up their right to free use of the lanes converted to Fast lanes. Credits, funded from toll revenues from the Fast lanes, could be used as toll payments on days when they choose to use the Fast lanes, or as payment for transit, paratransit, or parking at commuter park-and-ride lots (Smith, 2002). In spite of its innovative concept, there has not been implementation of the FAIR lane concept.

2.2.1.3 Cordon pricing

In Cordon pricing system, drivers must pay to enter and/or exit a whole city area or the inner parts of a city such as a Central Business District (CBD). This tool is mainly used in Europe (e.g., London, Norway, and Germany, etc.) and Asia (e.g., Singapore). One of the most recent implemented examples is a congestion charging system in central London on in 2003. It was introduced in an attempt to reduce traffic levels and help ease severely clogged roads. Vehicles entering central London within the Inner Ring Road have to pay £5 ($ 8) per day.

2.2.2 Pricing methods and systems

The transportation agencies have implemented a variety of pricing methods that range from simple pricing methods to more complex pricing methods and systems. These pricing methods have taken into account the time variability of pricing as well as the distance variability of pricing (WSA, 2002). The former takes into consideration time variance in terms of whether or not the pricing system changes with time. The latter of pricing options considers the distance variance in terms of whether the pricing system changes with distance. However, time variability of pricing options can be combined with any of the pricing concepts based on distance variability.

- **Time variability**

The three distinct pricing options with respect to time variance include pricing system with fixed schedule, preset variable rates, and dynamic variable pricing (WSA, 2002). The fixed schedule option has essentially no variability by time of day and is not based on levels of demand. Most traditional toll facilities follow this pattern. The pricing method with preset variable rates is varied by time of day and/or travel direction. Due to its fixed schedule preset, however, the pricing rates do not dynamically change with demand or level of service. The third option, dynamic variable pricing, is considered the most interesting pricing method. With this pricing option, toll rates vary based on actual levels of traffic demand, and not on a fixed schedule. The use of dynamic pricing can increase the ability to manage demand, and add complexity to both the toll system and its interface and communications with users. This is the approach currently in use on the existing I-15 HOT lane system.

- **Distance variability**

There are three basic toll rates with distance variability of pricing: flat tolls, per-mile tolls, and segment tolls (WSA, 2002). Flat tolls are a single rate, which is charged from any given point of entry regardless of a trip length. Per-mile tolls are the toll rate at any given time based on the distance traveled in the toll lanes. Segment tolls are charged per segment (i.e., the portion of the toll lanes between each pair of access points).

2.3 Value Pricing Examples in the U.S.

2.3.1 Variable tolls on toll facilities: LeeWay project in Lee County, Florida

LeeWay is Lee County’s Electronic Toll Collection system, installed on the Cape Coral and Midpoint bridges in November 1997 (LeeWay Service Center, 2009). The bridges are two of the four bridges that connect Cape Coral and Fort Meyers. The LeeWay project began in August 1998 on the bridges. However, at that time Lee County did not suffer from severe congestion on both bridges. This toll creation was beneficial to the study of variable pricing because any traffic changes resulting from variable pricing were primarily due to economic factors (i.e., toll savings) and not congestion (i.e., reduced travel time outside the peak period) related (Burris, 2001). These factors help to attribute observed traffic changes to the variable pricing program.

- **Pricing mechanism**

Pricing mechanism is unique in that they provide discounted preset toll pricing systems. For example, toll administrators provide a 50 % discount during the shoulder periods (6:30 to 7:00 am, 9:00 to 11:00 am, 2:00 to 4:00 pm, and 6:30 to 7:00 pm), in order to encourage drivers who usually travel during peak periods to change their time of travel (LeeWay Service Center, 2009). Before the start of the program, it was not a variable toll pricing system. In 1994, the Lee County Board of County Commissioners raised the tolls from $0.75 to $1.00 on the Cape Coral Bridge to help finance the construction of the Midpoint Bridge. At that time, the Board promised citizens that no further toll increases would be instituted in the foreseeable future (Burris and Swenson, 2001). Consequently, they proposed the current discounted variable pricing program, instead of a higher toll during the peak hours.

2.3.2 Value pricing: SR-91 Express Lanes in Orange County, California

The value pricing project of State Route 91 in Orange County, CA, called SR-91 Express Lanes, is the first application of value pricing in the U.S opened in December 1995, and the world's first fully-automated toll facility (SR-91 Express lanes, 2009). The SR-91 was originally HOV-3 or more facility. This is a unique project in terms of a public-private partnership project between Caltrans (California Department of Transportation) and a private company, California Private Transportation Company (CPTC). The SR-91 Express lanes consist of four lanes, 10-mile toll road built in the median of California's Riverside Freeway (SR 91) between the Orange/Riverside County line and the Costa Mesa Freeway

(SR 55). The toll lanes of two lanes in each direction are separated from the regular freeways by a soft barrier consisting of a painted buffer with pylons. The Express lanes do not have intermediate exits or entrances along the 10-mile length.

- **Pricing mechanism**

This project is the first operation under the term "value pricing," which literally means that rates are adjusted in proportion to the "value" drivers receive in terms of time savings versus the toll-free lanes. In SR 91 Express Lanes, tolls vary by time of day to reflect the level of congestion delay as well as to encourage travel during off-peak periods (SR-91 Express lanes, 2009). Tolls, however, follow a pre-determined, fixed toll schedule. As of November 1, 2001, tolls on the Express Lanes varied between \$1 and \$3.60 in the westbound direction and between \$1 and \$4.75 in the eastbound direction. However, to address when volumes approach levels at which free flow of traffic might be at risk, a new toll schedule is developed and published.

2.3.3 Value pricing: HOT lanes of I-15 FasTrack program in California

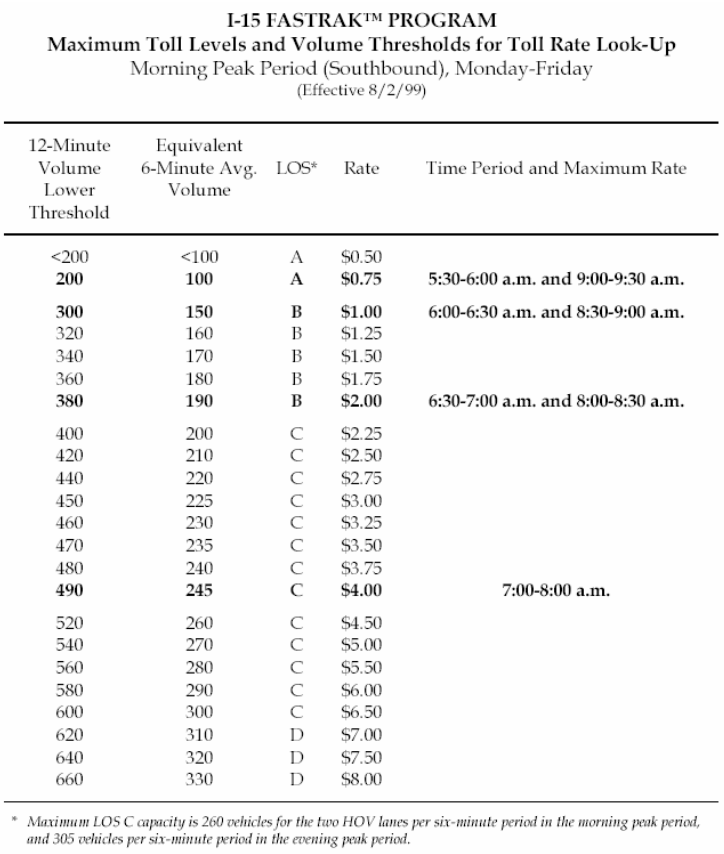
The Interstate 15 highway was originally opened as an HOV-2 facility in 1988, and then under the Value Pricing Pilot Program, the HOV lane was converted to a current HOT lane, named as a FasTrack Program, and implemented in 1996 (I-15 FasTrak Program, 2009). The HOT lanes are an eight-mile two-lane, reversible HOV facility in the median of Interstate 15. They are located in between the junction of I-15 and State Route (SR) 163 at the southern end and the I-15/SR 56 junction at the northern end. Concrete barriers from the regular traffic lanes physically separate the lanes.

Based on the success of the value pricing demonstration project (FasTrak Program), Caltrans and San Diego Association of Governments (SANDAG) are now planning to extend the project, called "Managed Lane" value pricing project (WSA, 2002). The primary objectives of the project are to extend the existing HOT lanes up to 20 mils to the north, and to widen the existing two-lane facility to four lanes. Instead of the fixed concrete barrier structured in the existing HOT lanes, a striped transition between the managed lanes and the general-purpose lanes is proposed. In addition, the new project will be designed with a movable barrier to permit unbalanced lane utilization during peak periods, with the two center lanes available for reversible flow operations. A movable barrier is to provide separation between opposing vehicles within the managed lanes. Another significant change from the FasTrak program is to design several new intermediate access points to and from the managed lanes. There will be generally three types of access. About seven transition areas will provide opportunities for transition between the general-purpose lanes and the managed lanes in each travel direction. The managed lanes are also designed to access directly to four planned bus rapid transit (BRT) and park-and-rides. There will be also about two other direct connections to local streets.

- **Pricing mechanism**

FasTrak program of I-15 has a unique feature of dynamic variable pricing; toll rates vary based on actual levels of traffic demand, and not on a fixed schedule. However, it is not a totally dynamic system. The current HOT lanes are physically operated like a closed highway; there is no intermediate access between the general purpose lanes and the HOT lanes. Drivers who want to access to the HOT lane facility are allowed only at its north and south ends.

On I-15 HOT lanes, there is a single "tolling zone" located in the southern half of the project area. Up to every six minutes, the loop detectors at this tolling zone continually measures total traffic (e.g., counts of vehicles equipped with transponders and traveling with a single occupant). Based on the real-time traffic levels read from the tolling zone, actual tolls are determined on the I-15 HOT lanes. Californian legislation required that the maximum total number of vehicles using the HOT lanes should not exceed 3,200 per hour to maintain Level of Service (LOS) of C level (SANDAG, 1999). Consequently, SANDAG decided that maximum LOS C capacity for HOT lanes is 260 vehicles for the two HOV lanes per six-minute period in the morning peak period, while 305 vehicles per six-minute period in the evening peak period. Because of the unbalanced capacity between traffics in the morning period for southbound toward San Diego, and those in the evening period for northbound, capacity of LOS C is slightly different in the southbound and northbound directions. SANDAG prepared simple look-up table with normal maximum rates for various time periods. An example of table for the morning peak period is shown in Figure 2.1. For instance, maximum target LOS C capacity is 310 vehicles per six-minute period for the two HOT lanes.



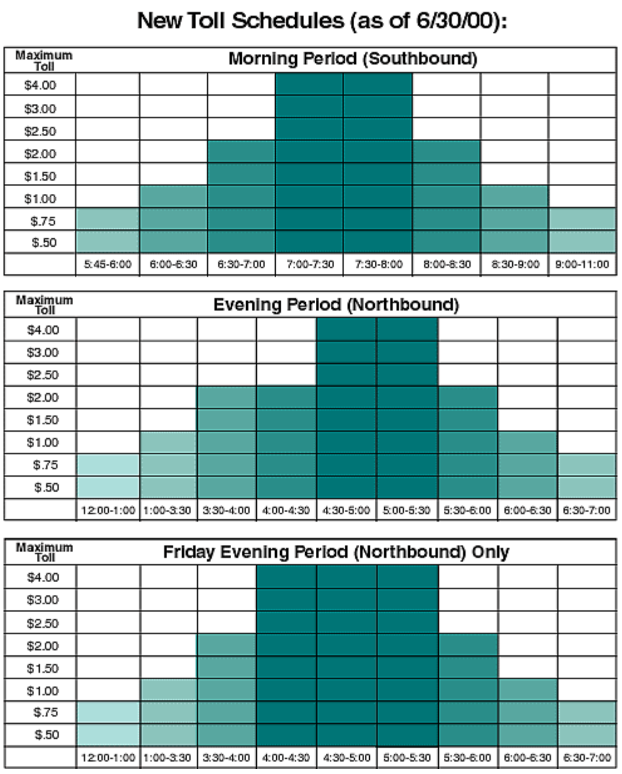
I-15 FASTRAK™ PROGRAM
Maximum Toll Levels and Volume Thresholds for Toll Rate Look-Up
Morning Peak Period (Southbound), Monday-Friday
(Effective 8/2/99)

12-Minute Volume Lower Threshold	Equivalent 6-Minute Avg. Volume	LOS*	Rate	Time Period and Maximum Rate
<200	<100	A	$0.50	
200	**100**	**A**	**$0.75**	**5:30-6:00 a.m. and 9:00-9:30 a.m.**
300	**150**	**B**	**$1.00**	**6:00-6:30 a.m. and 8:30-9:00 a.m.**
320	160	B	$1.25	
340	170	B	$1.50	
360	180	B	$1.75	
380	**190**	**B**	**$2.00**	**6:30-7:00 a.m. and 8:00-8:30 a.m.**
400	200	C	$2.25	
420	210	C	$2.50	
440	220	C	$2.75	
450	225	C	$3.00	
460	230	C	$3.25	
470	235	C	$3.50	
480	240	C	$3.75	
490	**245**	**C**	**$4.00**	**7:00-8:00 a.m.**
520	260	C	$4.50	
540	270	C	$5.00	
560	280	C	$5.50	
580	290	C	$6.00	
600	300	C	$6.50	
620	310	D	$7.00	
640	320	D	$7.50	
660	330	D	$8.00	

* *Maximum LOS C capacity is 260 vehicles for the two HOV lanes per six-minute period in the morning peak period, and 305 vehicles per six-minute period in the evening peak period.*

Figure 2.1 Look-up table for maximum toll levels (SANDAG, 1999).

The actual pricing is decided as follows. Every six minutes, data from the loop detectors at the tolling zone are read. Two six-minute traffic counts are added together, and then compared to a look-up table to determine the price. This rate is then displayed on the electronic toll display signs (SANDAG, 1999). The pricing is a per-trip based toll. Normal tolls charged vary from $0.50 to $4.00. Although the toll rates vary depending on the real-time traffic, SANDAG provides the maximum toll rate schedule that will normally be charged during a specified time period (Figure 2.2). Although the toll will not exceed this maximum rate for any time period shown on table (maximum toll rate is $4.00 in Figure 2.2), tolls could be higher as high as $8.00 if traffic on the HOT lanes is significantly congested approaching LOS C level. Usually, the actual rates charged in the morning peak on a typical day were below the maximum theoretical levels. On the other hand, due to higher demands, tolls charged in the evening peak on a typical day were at the theoretical maximum levels.



New Toll Schedules (as of 6/30/00):

Maximum Toll	Morning Period (Southbound)							
$4.00				🟩	🟩			
$3.00				🟩	🟩			
$2.50				🟩	🟩			
$2.00			🟩	🟩	🟩	🟩		
$1.50			🟩	🟩	🟩	🟩		
$1.00		🟩	🟩	🟩	🟩	🟩	🟩	
$.75	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
$.50	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
	5:45-6:00	6:00-6:30	6:30-7:00	7:00-7:30	7:30-8:00	8:00-8:30	8:30-9:00	9:00-11:00

Maximum Toll	Evening Period (Northbound)								
$4.00					🟩	🟩			
$3.00					🟩	🟩			
$2.50					🟩	🟩			
$2.00			🟩	🟩	🟩	🟩	🟩		
$1.50			🟩	🟩	🟩	🟩	🟩		
$1.00		🟩	🟩	🟩	🟩	🟩	🟩	🟩	
$.75	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
$.50	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
	12:00-1:00	1:00-3:30	3:30-4:00	4:00-4:30	4:30-5:00	5:00-5:30	5:30-6:00	6:00-6:30	6:30-7:00

Maximum Toll	Friday Evening Period (Northbound) Only								
$4.00				🟩	🟩	🟩			
$3.00				🟩	🟩	🟩			
$2.50				🟩	🟩	🟩			
$2.00			🟩	🟩	🟩	🟩	🟩		
$1.50			🟩	🟩	🟩	🟩	🟩		
$1.00		🟩	🟩	🟩	🟩	🟩	🟩	🟩	
$.75	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
$.50	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
	12:00-1:00	1:00-3:30	3:30-4:00	4:00-4:30	4:30-5:00	5:00-5:30	5:30-6:00	6:00-6:30	6:30-7:00

Figure 2.2 Maximum toll schedule in I-15 Value Pricing Pilot Project (SANDAG, 1999).

To prevent abrupt toll rate change, the toll rate is designed to increase or decrease gradually. The system is structured such that the price generally changes in $0.25 increments in any six-minute intervals. In other cases, during a typical morning or evening period, the toll will start at $0.50, increase to $4.00 during the peak, and then decline again as traffic drops off (SANDAG, 1999). It should be noted that while the toll rate is dynamically variable, only a single rate is displayed and charged at any given instant to drivers. As they approach the lanes, drivers will see the current toll rate displayed on variable message signs located at entry points. This toll rate is the actual price for the driver to pay. There are five electronic signs at both the north and south ends.

2.3.4 Value pricing: I-10 (Katy Freeway), QuickRide program in Houston, Texas

Interstate-10 freeway (Katy Freeway) was originally a 13-mile, reversible HOV lane first opened in 1984, flowing inbound in the morning and reversing in the afternoon (HHH, 2009). At this time only transit and vanpools were permitted on the lane. After expansion to include HOV-2+ vehicles, the lane was upgraded again to include only vehicles with three or more passengers during peak periods (6:45-8:00 a.m. and 5:00-6:00 p.m.) while HOV-2 + vehicles could access the lanes during the remainder of the day in order to maintain the quality and service of the lanes. However, the strategy only allowing HOV-3+ vehicles was not efficient, because of the problem of underutilization. In response to this problem, the Texas Department of Transportation, Houston Metro (Harris County Metropolitan Transit Authority) and FHWA introduced a value pricing project, called QuickRide program, on an existing HOV lane of the Katy Freeway and US 290. The QuickRide program, began in January 1998, is a smaller program relative to the other HOT lanes. The program was intended to utilize the excess capacity during peak periods without degrading the quality of the lanes. The program allows a limited number of HOV-2 vehicles, equipped with a transponder and a pre-paid account, to use the HOV-3+ lane during the peak hours for a fixed $2.00 fee for each trip. HOV-3+ vehicles can still use the HOT lane for free, while HOV-2 may use it without charge any time other than peak hours. However, to maintain existing levels of service on the HOV facility, the HOV-2 vehicles can be limited access, so that the target maximum number of QuickRide vehicles is 600 during each peak hour (Smith, 2002). Unlike other HOT lanes value pricing project, the QuickRide program does not allow single-occupant vehicles to use the HOT lane. The project is completely automated with the automated vehicle identification (AVI) technology.

2.3.5 Cordon pricing: Central London congestion charging

Central London congestion charging was introduced into central London on February, 2003 (London Congestion Charging, 2009). This is one of the most radical congestion charging schemes in big city. This experiment will significantly have impact on other cities including the U.S., Europe, as well as other major cities in the world. It is expected that the charge reduces congestion by up to 15% and raises at least £130 million a year which, by law, will have to go back into the capital's transport system. This charging system is a type of cordon tolls, although the charging zone is much greater than traditional cordon tolls. The congestion charging zone is 8 square miles within the Inner Ring Road; representing 1.3% of the total 617 sq miles of Greater London; 174 entry and exit boundary points around zone (London Congestion Charging, 2009). All drivers who enter the central London have to pay £5 (about $8.00), per day. There is no charge for traveling along the Inner Ring Road boundary itself. The charge is active between 7 a.m. and 6.30 p.m. Some vehicles are exempt from the charge such as taxis or emergency services. Drivers can pay the toll by telephone, text message, post, internet, or in person at a retail outlet. There are no tollbooths, barriers or tickets; instead drivers pay to register their vehicle number plates on a database without any equipment such as a transponder. About 900 cameras, situated at a network of 230 sites, inspect the number plates of vehicles entering or moving in the central zone (London Congestion Charging, 2009).

2.4 Equity Issues

Equity refers to the distribution of costs and benefits resulting from a policy decision. Because value pricing can produce large amounts of revenue, it can have a variety of equity impacts on various sectors of population. There are generally three types of equity (Litman, 1999).

- Horizontal equity
- Vertical equity with regard to income and social class
- Vertical equity with regard to mobility need and ability

A perfect assessment of the equity impacts on value pricing is not possible due to its complexity. This is part of the reason for the difficulty of defining "equity" itself. Litman

(1996 and 1999) pointed out several other controversial issues and difficulties of analyzing and evaluating equity. The assessment needs comparison of the proposed project with existing situation, requiring extensive data requirements. Equity analysis also depends on how groups are defined. In general, equity analysis of large groups is less accurate than if groups are disaggregated into smaller groups, or individuals. Vertical equity is often a debatable issue. People who support vertical equity argue that society is defined morally according to how it treats disadvantaged members, while others point out that providing extra resources to the disadvantaged population reduces the incentive for individuals to overcome such disadvantages. In addition, vertical equity with regard to income and social class, which will be described below in detail, is not perfect metric because people with the same income often have very different needs and abilities. In spite of the weakness, lots of analyses focused on the equity issue under this definition to evaluate broad equity impacts, because income data is easily obtained. Due to these difficulties, most current analyses of transportation equity focus on only one of three equity types.

2.4.1 Horizontal equity

Horizontal equity, called also fairness, refers to whether people are treated equally. It implies that consumers should get what they pay for, and pay for what they get, unless there is a specific reason to do otherwise (Litman, 1996). However, who deserves the benefit according to this criterion is a matter of debate (Litman, 1999). It can be defined as just those who actually pay the toll, or it could also include those who change their travel patterns in response to the toll, thereby incurring costs in terms of inconvenience, and providing the congestion reduction benefit to the toll payers. Road pricing is often criticized on the grounds that they represent the transfer of public resources to benefit one group of citizens at others expense (VTPI, 2009). Congestion pricing benefits higher income motorists who have a high ability to pay, and it can increase congestion delays to other road users. Some critics argue that road pricing is horizontally inequitable because it represents double taxation; motorists already pay for highways through fuel taxes and other roadway user fees. However, roadway user fees do not cover the full costs of roadway use, which leave other additional uncompensated external costs. Therefore, the additional road user fees can be justified on horizontal equity grounds. The horizontal inequity can be offset if some of the benefits of road pricing are distributed to lower-income motorists and non-drivers by investment or subsidies to the public transportation for improvements. For instance, a subsidy to reduce transit fares is a simple alternative way for non-drivers to receive their share of transportation resources.

2.4.2 Vertical equity with regard to income and social class

This type of equity focuses on the allocation of costs between income and social classes (Litman, 1999). From this definition, road pricing is most equitable if it provides the greatest benefit at the least cost to disadvantaged groups, therefore compensating for overall social inequity (Litman, 1999). Value pricing is considered vertically inequitable if toll pricing imposes a larger burden on the disadvantaged groups. Giuliano (1994) argues that women commuters would bear an unfair financial burden from road pricing because they tend to have fewer travel options, due to family responsibilities and inflexible employment conditions, although this would be offset for higher-income women by reduced congestion delays. However, the regressivity of pricing depends on the travel options available to disadvantaged groups and how revenues are used (Litman, 1996). Road pricing is vertically inequitable when the distribution of revenues is ignored, but becomes beneficial to all income classes if revenues are either returned proportionally to each class or distributed equally per capita (Small, 1992b). Many other publications also present the positive sides of vertical equity issue. Kain (1994) identifies significant potential benefits to lower income commuters and non-drivers from congestion pricing because it could improve transit and ride sharing services. Johnston et al. (1995) considers the income equity of congestion management strategies, including HOV facilities, metering, pricing, and rationing. They conclude that pricing can be equitable if revenues are appropriately spent. However, few publications were made in terms of equity issues with the currently operated value pricing project because of its difficulty of evaluation. One analysis conducted by Burris and Hannay (2003) is the study of the impact and use of QuickRide HOT lane project on the Katy Freeway in Houston, Texas. The study focused on vertical equity issues with regard to income and social class, by examining primarily by income level and by occupation with statistical analysis. The analysis used a combined dataset of revealed preference survey data with data on actual HOT lane use. Accordingly to the study, QuickRide usage did not vary significantly respondent's income, occupation, age, or household size. The study also showed that HOT lanes users were found to have significantly higher incomes and to be significantly younger than drivers on the free-lanes. However, it was also found that no drivers were made worse off due to the program.

2.4.3 Vertical equity with regard to mobility need and ability

This definition assumes that everyone should enjoy at least a basic level of access, even if people with special needs require extra resources. Applying this concept, however, can be difficult, because there are currently no standards for transport need, or a consistent way to measure access (Litman, 1999). There are no detailed analyses from the real practices of value pricing system regarding vertical equity with regard to mobility need and ability yet.

2.5 Enforcement and Monitoring/Evaluation

2.5.1 Enforcement

Enforcement and compliance can be effectively accomplished by the combination of surveillance systems and Highway Patrol activity. In particular, new technologies, such as ETC system, have made this enforcement easier. For example, SR 91 HOT lanes and I-15 HOT lane projects use a photo-enforcement technology. SR 91's express lanes have dramatically decreased congestion along one of the state's busiest highways by using AVI and advanced traffic management systems (ATMS). The 10-mile stretch of no-cash toll road is the first in the U.S. with a congestion-based pricing system. Overhead antennas read radio signals from transponders inside motorists' cars, and the appropriate toll is deducted from their pre-paid account. Closed-circuit television cameras monitor the toll road and photograph the rear license plates of all toll violators. Citations are sent via mail. Like SR-91 system, in I-15 HOT lanes, the video cameras installed on gantries over the toll zone monitor and provide a record of violators on HOT lanes (SANDAG, 2001).

Another good example of enforcement system is the case of central London Congestion charging. The enforcement of the £5 charge and penalty notices is being undertaken by making use of a massive CCTV monitoring system as well as Automatic Number Plate Recognition (ANPR) technology (London Congestion Charging, 2009). When motorists enter the charging zone, road signs alert them. Every single lane of traffic is monitored at both exit and entry points to the charging zone. No tollbooths are required because drivers pay to register their vehicle number plates on a database. The plate number of vehicles entering in the central zone is then inspected by a network of fixed and mobile cameras. About 900 cameras are installed at a network of 230 sites, of which just fewer than 180 are on the Inner Ring Road. The cameras provide high quality video-stream signals, and the captured images are then sent back to the ANPR computer system. The ANPR stores data showing the exact time, and date that the images were taken. All images are then automatically matched against a database of those who have registered to pay. Vehicles without valid permits are traced through the Driver and Vehicle Licensing Agency to find the registered keeper, who is then fined. Tests showed that there is an estimated capture of 90 % within the charging zone.

Although Intelligent Transportation System (ITS) technologies such as ETC system have proven very popular among drivers, it is important to remember that the public still concern about an invasion of privacy. Therefore, transportation agencies responsible for the value pricing project must address privacy issues on how personal privacy is protected with the systems through public outreach efforts or programs.

2.5.2 Evaluation and assessment

In most cases, data for evaluation and assessment are obtained from two primary sources: the field observation data carried for quantitative traffic studies and surveys for qualitative studies of travel behavior. National Research Council (1994) suggested six critical questions as well as some other issues for evaluation of value pricing. The major questions include:

1. What is the range of behavioral responses at different prices?
2. How will behavioral changes affect congestion?
3. What will the impacts be on different groups (considered by income, gender, and geographic area)?

4. What will the air quality and energy effects be as a result of value pricing?
5. What will the effects on urban form and development be?
6. How do all of the above affect public receptivity and political feasibility?

This section compares evaluation guidelines suggested by NRC with those evaluated in the current value pricing projects. A simple comparison of monitoring and evaluation is presented in Table 2.1, based on evaluation criteria suggested by NRC.

- **Travel behavior**

Travel behavior analysis is a particularly important area for evaluation. Questions would be about how personal, business, and commute trips are affected by value pricing program. All current projects selected above provide this analysis. Travel behavior data for measurement are generally collected by user surveys, focus group results, or telephone surveys.

Table 2.1 Comparison on Evaluation Topics (Sources from NRC, 1994; SANDAG, 2001; Burris, 2001; Sullivan, 2000; Transport for London, 2003)

	NRC	I-15 HOT lane	Lee County variable pricing	SR-91 HOT lane	London charging system
Key evaluation	Travel behavior	Attitudinal panel study (characteristics of travel behavior, perceptions of traffic conditions, and attitudes)	User surveys, focus group results, telephone surveys	Travel behavior	Travel behavior
	Effects on facilities	Traffic study (traffic volumes, vehicle occupancy and classification, and speed/delay, park and ride study)	Traffic study (vehicle occupancies, volumes, crashes, erratic maneuvers at the toll plazas, speeds, travel time, vehicle queue lengths)	Traffic trends, toll/non-toll facility shares, amount of trip making and trip Purpose, travel times and perceived travel conditions, transponder acquisition and use	Congestion, traffic counts, secondary transport effects (parking, London taxis/minicabs, pedestrians, motorcycle and pedal cycle activity)
	Impacts on specific groups			Travel choice and elasticities**,** income and other demographics	Economic and business impacts, social impacts
	Environmental and energy consequences	Air quality study (VOC, NOx, particulate matter less than 10 microns (PM10), CO)		Vehicle emissions	Environmental impacts (air quality, energy use, greenhouse gas emissions, noise)
	Effects on urban form				
	Political and public receptivity	Media relations and coverage, marketing, and public response	Marketing and public awareness efforts	Public opinion	
Other evaluation	Commercial transportation and activities	Business impact study		Employer trip reduction Programs	
	Impacts on transit and other modes	Bus rider-ship	Transit rider-ship	Ridesharing and transit	Public transport
	Measures of congestion				
	Land use changes	Land use study			
	Shifting of burdens				
	Regional politics				
Unique evaluation		Cost of delay study (value of time), enforcement effectiveness and violation assessment		Collision trends and characteristics	Scheme operations (the payment channels used; and the revenues raised)

- **Effects on facilities**

This is to assess whether and how much value pricing reduces congestion on specific facilities. Traffic studies, such as traffic volumes, vehicle occupancy and classification, and speed/delay, are major categories of the study. Impacts on transit (e.g., transit rider-ship) can be included in this study. For instance, SR-91 value pricing project evaluate the impacts of value pricing project on travel patterns by using statistical analysis (Sullivan, 2000). They used Chi-square hypothesis test or standard linear regression analysis for the difference. However, special carefulness is required to assess this kind of effects, due to external effects on travel behavior such as seasonal, economic effects on travel, or other factors. For this reason, evaluation of I-15 HOT lane project identifies and takes into consideration external factors (e.g., gasoline prices, employment levels, interest rates, etc) (SANDAG, 2001). To control those effects, the study established a control corridor (I-8) to distinguish changes attributes to the I-15 value pricing project from changes due to variables outside the project.

- **Impacts on specific groups**

This analysis encompasses economic analysis, demographic data, and equity issues of value pricing, using data collected by user or telephone surveys. The unique analysis was performed by Resource Systems Group (RSG) and Wilbur Smith Associates for SR-91 HOT lane value pricing project (Sullivan, 2000). Using the travel survey data, the study calibrated travel demand of route choice models and price elasticities. In particular, the study performed several different model specifications to test, from the simplest test of price elasticity of toll road use with route choice only, using the logit model, to the expanded models to allow for the route choice to be made simultaneously with the other choices such as models of choice of mode, time of day, transponder, using the joint logit model as well as a nested logit model. Similar studies were performed in other projects also. In LeeWay project, Center for Urban Transportation Research, as a primary team for monitoring and evaluation, performed statistical tests to compare differences between prior to the start of variable tolls and after the introduction of variable tolling system (Burris, 2001).

- **Environmental and energy consequences**

Because there are more and more concerns about the environment and energy conservation, it is important to collect data on both travel behavior and actual emissions.

- **Effects on urban form**

Value pricing may affect urban form. However, due to its difficulty of measuring impact on urban form, no current value pricing projects provide this type of analysis.

- **Political and public receptivity**

The public perception, response, and awareness on value pricing are very important measurement for evaluation. This issue sometimes determines whether value pricing project is successful or not. Specific attention should be given to sample design to ensure that it captures those most affected (for good and ill), but also to ensure that the sample in the aggregate representative of users of the facility (NRC, 1994).

Chapter 3. Literature Reviews

3.1 Literature Review on Value Pricing Modeling and Simulation

Value pricing has been studied extensively by both transportation researchers and economists. For a long time, economists presented a variety of congestion pricing models with different modeling assumptions. Using the traditional microeconomic theory of marginal-cost principle, they primarily focused on the static situation. On the other hand, the research on dynamic congestion pricing in a network is not widely studied, and when it is studied, it is often restricted to hypothetical, idealized situations. This is mainly due to complications of the network structure, analytical models, and algorithms used in the study of dynamic value pricing. Value pricing analyses should be able to capture and reflect the dynamic processes of traffic, transportation networks, and demand in terms of both time and space, by considering time-dependent toll pricing scheme, for instance.

The congestion pricing models have widely been studied from different modeling perspectives with diverse modeling assumptions. The earlier studies on congestion pricing can be classified into three types: 1) static space dimension, 2) time dimension, and 3) joint of the time and space dimensions (Yang and Huang, 1997). The first approach, static space dimension, is mainly studied based on economic theory (Yang, 1999; Yang and Huang, 1998, 2004; Yang and Lam, 1996; Yin and Yang, 2004). These studies used the traditional marginal-cost pricing theory, where a toll charged is equal to the difference between marginal social and marginal private costs. In the second approach of the time dimension, various time-varying charging methods have been examined. However, this type of study is limited to hypothetical and idealized situations, considering only a single queuing bottleneck (Vickrey, 1969; Arnott et al., 1993; Ben-Akiva et al., 1986). The third type of approach combines both dimensions of time and space for the analysis of congestion pricing. These studies try to capture the dynamic situations, although the studies involve more complications and complexity of dynamic traffic assignment and network structure (Arnott et al., 1990; Carey and Srinivasan, 1993; Yang and Meng, 1998).

A few problems identified in the analytical models could be solved using a simulation-based approach. For example, Abdelghany et al. (2000) and Murray et al. (2000) used the dynamic traffic assignment (DTA) mesoscopic simulation model, DYNASMART (Dynamic Network Assignment simulation Model). Kwon et al. (2000) combined analytical DTA and PARAMICS microscopic simulation model to study the impacts of different pricing schemes. In this study, however, they failed to account for heterogeneity of travelers in terms of multi-class and/or VOT. One of the most recent studies conducted by De Palma et al. (2005) used METROPOLIS simulation model, but they assumed constant VOT for all travelers. Their analysis was also restricted to home-to-work trips. The inability to consider heterogeneity of travelers is inherent in those simulation models, since they use aggregate origin-destination (OD) data and do not have the ability to consider individual socio-economic information.

Apart from this distinction based on time-space dimension and microsimulation-based approach, the researchers have made an attempt to formulate mathematical programming to solve the congestion pricing problem using different modeling approaches. These approaches include: 1) the marginal cost pricing as a mathematical formulation/programming, optimization problem based on the fundamental marginal cost theory, 2) bi-level programming, 3) variational inequality approach, and 4) optimal control theory. The following four sections provide the literature reviews on these analytical approaches to model and analyze congestion pricing.

3.1.1 Value pricing model based on marginal-cost theory

Many economists and transportation researchers have tried to formulate and analyze the efficient congestion pricing that is based on the fundamental economic principle of marginal-cost pricing (Carey and Srinivasan, 1993; Yang and Huang, 1998; Liu and McDonald, 1998, 1999; and Verhoef, 2000). These studies state that a road user using a congested road should pay a toll equal to the difference between the marginal social cost and the marginal private cost in order to maximize social net benefit. The marginal congestion cost, which is now equal to the toll, is known as an externality. This is the externality in the economic sense, since each additional user driving on a congested road imposes a congestion cost on other road users, on the other hand, the traveler takes into account only his/her own costs in deciding , when, or which route he/she takes, and does not consider the costs which he/she imposes on others. A social optimum can be obtained by imposing a toll equivalent to the externality. By internalizing the user externality through the imposition of the optimal congestion toll, a system optimum can be achieved.

An abundance of research has been conducted using the theory of marginal cost pricing with a variety of different modeling assumptions; for instance, pricing method (first-best, second-best); toll rates (uniform/dynamic/peak and off-peak); network (bottleneck, two-routes, general network); demand (fixed, elastic); user reaction model (route choice only, both route and departure time choices); and user classes (single and homogenous, multi-class). In contrast to the works modeled with uniform toll rates, dynamic congestion pricing model based on the marginal cost pricing was studied by Arnott et al. (1990), de Palma and Lindsey (1997), and Carey and Srinivasan (1993). Taking into account both departure time and route choice models, Arnott et al. (1990) analyzed various pricing policies (uniform, step tolls, and time-varying). However, their model is limited to either a bottleneck or a single destination network and used for a fixed demand. Yang and Huang (1998) investigated the principle of marginal-cost pricing in a general congested road with queuing. While most of the models based on the marginal cost pricing were developed using static user equilibrium, Yang (1999) modeled in a congested network in a stochastic equilibrium. Recently, analysis of marginal cost pricing with multiple user classes are conducted by Yang and Huang (2004), Yin and Yang (2004), and Zhao and Kockelman (2004).

It should be noted that most early studies assumed that tolls are levied on every link of a network, called "first-best" pricing. However, this assumption is not possible or unrealistic due to technical and/or political constraints. As a result of this limitation, the theory of "second-best" pricing has been investigated by several researchers. Under the second-best pricing scheme, the tolls are charged only on a given subset of links of the networks. A simple example of the second-best pricing is an analysis with two parallel routes where both a tolled and a non-toll alternative exist. In recent years, there have been many researchers studying on this topic (Braid, 1996; Liu and McDonald, 1998, 1999; de Palma and Lindsey, 2000; Verhoef, 2000, 2002a, 2002b).

3.1.2 Value pricing model using bi-level optimization problem

Some researchers tried to solve congestion pricing model based on game theory (Abou-Zeid and Chabini, 2003; Yang and Bell, 1997; Yang and Lam, 1996; Viti et al., 2003). Game theory can be applied to model interactions between groups of decision-makers when individual actions jointly determine the outcome (Fisk, 1983). The bi-level programming model for solving the congestion pricing problem is motivated by the fact that there are two-players (in other words, a leader and followers) or two-decision-makers, known as a Stackelberg game in game theory. In Stackelberg game, the leader goes first and attempts to minimize costs, taking into account all possible responses of the followers (Bard, 1998). The followers observe the leader's decision and react to optimize their objectives. In a bi-level problem equivalent to the Stackelberg game, the leader (e.g., transportation agency) sets and implements the tolls to optimize his objective, for instance, to minimize total travel cost in the whole transportation networks. The followers (e.g., the network users) choose their routes to achieve their objectives, for instance, travelers choosing a route based on travel time and/or cost. Accordingly, a bi-level toll problem has two areas to consider: upper level problem (a transportation planner's problem) and lower level problem (a network users' problem). Figure 3.1 shows the interactions between the two levels (Viti et al., 2003). The leader (transportation agency) in the upper level sets and adjusts iteratively the toll level to find out travel pattern that minimizes the total travel cost. Thus, the upper level can be structured as the optimization problem in terms of total system costs. The lower level can be structured as the user equilibrium model, in which each network user (follower) chooses the route that minimized his/her travel costs, given the toll setting and the demand pattern. As a result of the (dynamic) traffic assignment, the equilibrated traffic flows and travel costs is fed back into the upper level.

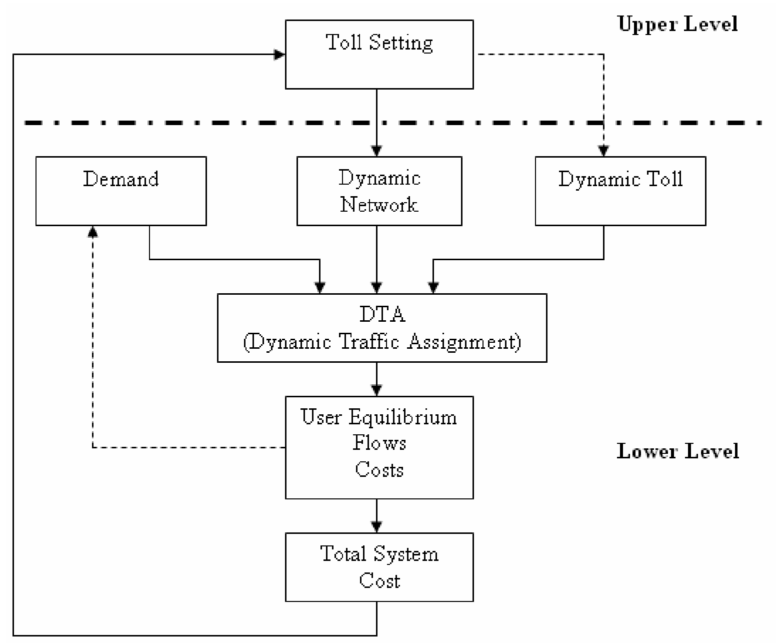


Figure 3.1 An example of bi-level program structure (Viti et al., 2003).

A bi-level model for solving congestion pricing formulated by Yang and Lam (1996) is to evaluate the optimal second-best tolls under conditions of both queuing and congestion. The optimal tolls in this algorithm are the second-best tolls where not all links of the network can be tolled. However, the model deals with steady-state traffic conditions by assuming that the OD demands are given and fixed. Yang and Bell (1997) introduced the elastic demand network equilibrium model with queue, by applying bi-level programming model. Yang and Zhang (2002) proposed a multi-class network toll problem using bi-level programming model with social and spatial equity constraints. In contrast to the static analysis, dynamic congestion pricing models based on a bi-level programming are proposed by Abou-Zeid and Chabini (2003) and Viti et al. (2003), making use of a dynamic traffic assignment model in the lower level problem. Viti et al. (2003) also introduced a departure time choice model as well as a route choice model into a lower level optimization of the bi-level problem. Recently, Chen and Bernstein (2004) developed a single-level, standard nonlinear optimization problem with multiple user groups. They converted a traditional bi-level formulation into a single level problem to improve efficiency of solution.

It should be noted that it is not an easy task to solve a bi-level programming for congestion pricing model due to the non-differentiability and non-convexity problems of objectives and/or equilibrium constraints. If modelers want to design a dynamic toll pricing using a bi-level programming, they should consider and solve a dynamic traffic assignment in a lower level of the bi-level model. Including dynamic transportation models can represent real situations better than the static models in the congestion pricing system. However, finding the solution to dynamic models is more complicated and difficult to prove the uniqueness of the solution due to the dynamics or the asymmetric interactions in the lower level problem. Thus, in the future, more studies are expected to develop an efficient solution algorithm of the bi-level programming approach in the congestion pricing design.

3.1.3 Value pricing model using variational inequality approach

Variational inequality (VI) problem, which is also known as the generalized equation and stationary point problem, was originally developed for the study of partial differential equations in mechanics, which is defined over infinite-dimensional spaces (Chen, 1999, and Nagurney, 1993). The VI problem contains a general problem formulation in such problems as nonlinear equations, optimization problems, complementary problems, and fixed point problems. This approach is widely applied in modeling various static and dynamic problems because of its relation to an equivalent optimization formulation, simple proof of the uniqueness for the VI problem, and the usefulness in applying to the optimization problem with the asymmetry condition. Compared to mathematical programming and optimal control approaches, the most useful portion of the VI formulation in the application of transportation problems is its capability in representing various dynamic situations, for instance, interactions among link flows on link travel times (Chen, 1999).

The earlier researches were conducted by Smith (1979) and Dafermos (1980). Smith (1979) is the first person who studied a static user-optimal route choice problem as

VI problem. Other research using VI approach in various transportation problems can be found in Fisk and Boyce (1983), Nagurney (1993), Friesz et al. (1993), and Ran et al. (1996). The studies on congestion pricing models were conducted by Liu and Boyce (2002), Viti et al. (2003), Yin and Yang (2004) and He et al. (2003). Liu and Boyce (2002) investigated the problem of optimal congestion pricing in a general transportation network with multiple time periods for the analysis of the temporal effects of congestion pricing. They presented a VI formulation of the system-optimal problem and derived a formula for calculating efficient congestion tolls, under both link toll and route toll policies. Viti et al. (2003) used a route-based VI formulation for their route choice problem in the dynamic congestion pricing model, which is solved using bi-level approach. He et al. (2003) proposed an analytical stochastic and dynamic network model formulated as VI problem for the multi-class dynamic network model. They evaluated various toll schemes for HOT lane design and operations. Recently, Yin and Yang (2004) studied the multi-class bi-criteria network equilibrium problem for the feasibility of marginal-cost pricing principle along with VI formulation.

3.1.4 Value pricing model using optimal control theory

In contrast to the mathematical programming approach, which is typically formulated as a discrete-time problem, the continuous optimal control theory can be used for the time-dependent traffic assignment problems and the corresponding dynamic congestion pricing model through modeling vehicle dynamics. The constraints in the optimal control formulation are defined in a continuous time set. That is, in a typical optimal control formulation, the OD trip rates and the link flows are assumed to be continuous functions of time.

The early studies on time-dependent traffic assignment problems using optimal control theory in either user-optimal or system-optimal were conducted by several researchers. Agnew (1977) applied optimal control theory to find optimal time-varying congestion tolls for a bottleneck. However, he used a complicated speed-flow-density relationship in the model. Friesz et al. (1989) proposed two continuous time formulations for the dynamic traffic assignment problems, including static system optimal model and dynamic user optimal model. The continuous time optimal control formulations developed by Wie et al. (1994) and Lam and Huang (1995) are a dynamic user equilibrium assignment. Wie et al. (1994) developed and presented an augmented Lagrangian method for solving dynamic traffic assignment models formulated as optimal control problems. Wie (1998) formulated the convex control problem for a dynamic system-optimal traffic assignment model with multiple origins and destinations. In the model, he improved previous formulations by eliminating the problem of computational difficulty in the non-convexity as well as prohibiting instantaneous flow propagation.

On the other hand, there are few studies on dynamic congestion pricing using optimal control theory. Yang and Huang (1997) proposed a time-varying congestion pricing model using continuous time optimal control formulation and differential game theory. The model is for a bottleneck analysis with elastic demand, queuing delay, and both variable and constant exit capacity of the bottleneck. Rather than the objective function of minimizing individual travel cost, this model seeks to maximize the net economic benefit. The time-varying congestion toll in this model is based on user externalities that are present in the optimality conditions. Wie (1997) developed a dynamic congestion pricing model formulated by a convex optimal control problem of the dynamic system optimal traffic assignment. Compared to the previous models of Friesz et al. (1989) and Wie et al. (1994), the proposed differential equations prohibit instantaneous flow propagation by dividing links as an un-congested segment with a constant free-flow travel time and a congested segment with a bottleneck with vertical queues. However, this dynamic congestion pricing model assumes a fixed travel demand. The most recent time-varying congestion model using an optimal control theory is found by Chang and Cheng (2003). This dynamic system optimal traffic assignment model is better than the previous models, by taking into account alternative mode, departure time, and route choice decisions in the optimal control problem. For the analysis of HOV lanes, they used two alternative modes: driving alone and carpooling. Each mode is assigned to a specific link. The model was analyzed with two link segments: an un-congested segment and a congested segment. The dynamic marginal travel costs were obtained from the derivation of optimality of the model. A time-varying congestion toll was based on the congestion externality.

3.2 Literature Reviews on Value of Time (VOT)

3.2.1 Background

There is extensive literature surrounding VOT for estimation in welfare economics, originally based on time allocation theories, as well as transportation planning field. In the field of transportation planning, VOT measures have frequently been used for the purpose of the economic evaluation of road infrastructure and public transport projects. It is commonly measured using discrete choice models, such as mode choice (e.g., car, bus, and rail) or route choice models, based on random utility theory (Ben-Akiva and Lerman, 1985). VOT measures have also been applied for the purpose of forecasting studies, allowing time and cost to be expressed in common units in the generalized cost formulation.

One of the significant weaknesses in most of traditional utility models is that VOT is assumed as a constant marginal VOT. This is partly due to inconsistency of experiment results or to the concerns of equity issues (Ortuzar and Willumsen, 2001). However, this assumption is not proper in reality and especially in the analysis of a HOT lane system. In a HOT lane system, VOT of each traveler plays a substantial role by influencing users' reaction on whether he/she will use the HOT lane by paying a toll (Yang et al., 2002). Every individual has his/her own VOT based on different preferences. Many researchers have reached the conclusion that individuals react differently to pricing strategies based on their VOT and based on their socio-economic characteristics (i.e., income) and their trip characteristics (i.e., trip purpose, travel time, travel cost) (Ghosh, 2000; De Palma and Lindsey, 2004; Small and Yan, 2001; Steimetz and Brownstone, 2005).

Even though many literatures on VOT have been published, only parts of them have focused on differentiating VOT. Moreover, an estimation of VOT in the context of a HOT lane value pricing is very recent. Thus, of what we are interested in this dissertation are such questions as: 1) which socio-economic variables or travel characteristics should be included in order to capture users' heterogeneity in estimation of VOT; and 2) what kind of relationships are there between socio-economic characteristics and level of service (LOS) variables (i.e., travel time and travel cost).

The following three sections illustrate how VOT is derived from a discrete choice model, describe the conventional assumption of constant VOT and several attempts to relax the assumption, and summarize various models with non-linear utility functions from the earlier papers.

3.2.2 How VOT is derived from a discrete choice model

A single, constant VOT is derived from a discrete choice model based on random utility theory. It is also referred to a point estimate in some literatures, since it is measured from the mean of travel time divided by the mean of the travel cost. Let U_{ij} be the conditional indirect utility function of alternative j for the individual i as

$$U_{ij} = \beta_c C_{ij} + \beta_t T_{ij} + \varepsilon_{ij} \qquad (3.1)$$

where, travel time, T and travel cost, C are observable characteristics, and the unobservable error term, ε_{ij}

VOT is the monetary value of the marginal rate of substitution between travel time, T and travel cost, C. It is also commonly referred to as consumer's willingness to pay to save a marginal unit of travel time while keeping the same level of utility. It is expressed as the ratio between the two estimated parameters of the attributes travel time (β_t) and cost (β_c) as shown in Eq. 3.2.

$$VOT = \frac{\partial U_i / \partial T_i}{\partial U_i / \partial C_i} = \frac{\beta_t}{\beta_c} \qquad (3.2)$$

Based on the traditional assumption of linear and additive in the fixed marginal utility parameters, when we take into account wage rate (or income), w , VOT as a percentage of income is now simply

$$VOT = \frac{w\beta_t}{\beta_c} \tag{3.3}$$

However, it has been recognized that individual user chooses an alternative in indirect utility function, based on the observable alternative specific characteristics as well as socio-economic and trip characteristics. The typical way of considering socio-economic variables in the utility is to include them as a linear function in the parameters and attributes, yielding the following utility function.

$$U_i = \theta_i + \beta_c C_i + \beta_t T_i + \beta_i X_i + \varepsilon_i \tag{3.4}$$

In the above utility function, X_i represents socio-economic variables (e.g., income, age, etc.).

3.2.3 Constant VOT and approaches to relax the assumption

The most popular method to measure VOT is using various forms of choice models (e.g., multinomial, nested, or mixed logit choice models) from either revealed preference (RP) or stated preference (SP) data. In the traditional approach, VOT is derived using the coefficients of travel cost and travel time as discussed in the previous section. However, the most significant limitation in the conventional VOT estimation is the assumption of the constant VOT over the population. Most of earlier econometric models were based on the assumption of homogeneous marginal VOT. However, it is not intuitive that VOT is constant in all situations. It ignores individual heterogeneity, thus results in biased estimates (Ghosh, 2000; Hensher and Goodwin, 2004; Small, 1992a). In addition, constant

VOT is not valid anymore in such a case as a HOT lane value pricing model, where single-occupant drivers choose the HOT lane, depending on their VOTs (willingness-to-pay).

Some researchers have tried to capture socio-economic variables into the random parameters of the utility function, still resulting in a constant VOT, as shown in Eq. 3.4. This approach may be applicable in such cases when individual information is not available, when it is difficult to measure or observe different users' characteristics, or when the effect of socio-economic characteristics on the utility function is yet inconclusive in a specific context. One disadvantage of this approach, however, is that although the utility function in a discrete choice model may include socio-economic characteristics and travel characteristics to reflect variability of individual in choosing an alternative, VOT estimated from this model cannot reflect heterogeneity of VOT for individual.

In order to reflect heterogeneity in VOT, some researchers have proposed VOT as a continuous random variable distributed across the population of travelers, following a certain probability density function (PDF) (Armstrong et al., 2001; Ortuzar and Willumsen, 2001; Fosgerau, 2004; Mahmassani et al., 2005; Romero et al., 2005). The effects of travel time and travel costs are captured in the deterministic part in this approach. With an insufficient data or observation, it may be acceptable to assume the effect of socio-economic variables reflected with unknown PDF given a certain level of confidence. However, most of socio-economic variables are measurable characteristics these days. As we continuously obtain much more detailed data and lots of diverse survey data available, the assumption of constant VOT is not useful approach anymore. For example, more transportation planners and engineers are increasingly attracted to the activity-based model, instead of popularly used models with aggregate data. TRANSIMS is a good example of activity-based simulation model. It deals with great amount of input data, including household as well as individual data. Some of input data include census data, vehicle ownership, gender, age, employment status, income, and trip purpose.

In order to overcome these problems, alternative approaches were introduced. The use of market segmentation approach is the simplest way to approximate VOT for each individual or by dividing the whole population into a number of groups (e.g., by trip purposes) (Mackie et al., 2003; and Whelan and Bates, 2001). A common segmentation is based on different trip purposes (e.g., business, commuting, and other trips). The segmentation is sometimes extended to account for income levels (e.g., low-, medium-, and high-income groups) or trip length. Along with segmentation approach, the most common way of introducing socio-economic variables into the systematic utility is to add them either linearly or in interaction with LOS variables to capture the differences in preference (Cherchi and Ortuzar, 2003; and Ortuzar and Willumsen, 2001). However, in the linear relationships of socio-economic variables with LOS variables in the utility function, socio-economic variables do not affect individual VOT estimation. In other words, socio-economic variables certainly influence the total utility associated with each alternative, however resulting in a fixed VOT value over the population.

In order to obtain heterogeneous VOT for each individual, socio-economic variables should be included as second order terms (i.e., non-linear relationships between socio-economic variables and LOS variables) in a utility function. Unfortunately, this subject is regarded as somewhat difficulty, and conclusive results among researchers do not exist at the present time. The only variable accepted from most researchers as a conclusive variable that is affecting individual VOT is income. Some socio-economic variables in the indirect utility formulation may not strongly be supported. However, most modelers agree with socio-economic and trip characteristics should influence VOT estimation in some way in order to fully capture heterogeneous VOT. The following section presents the earlier findings for the variability of VOT among individuals with a non-linear systematic function between LOS variables and measurable characteristics of socio-economic variables.

3.2.4 Empirical results of VOT with non-linear functions

Various empirical studies have suggested that VOT varies significantly across individuals by different trip characteristics, including trip purpose, mode and size of time savings, as well as personal attributes such as income. A number of representative results are summarized in the studies of Brownstone and Small (2005), Konig et al. (2003), Mackie et al. (2003) and Wardman (1998), although most of them deal with the traditional utility models with the constant VOT assumption. The reasonable estimated VOT from several studies is about 50 percent of gross wage rate (Miller, 1989). This assumption has been frequently accepted and used in many applications, although it is more or less practical approximation rather than stable relationship. Table 3.1 presents nine papers that utilize heterogeneous VOT and all variables used in each model.

The model of Small et al. (1999) shows the effect of income on VOT estimation. They developed five different models to investigate the effect of congestion on the value of travel time using the SP survey data collected from SR-91 users in California. Using logit choice model, they found that income is the only variable that has much effect on the valuation of total travel time, whereas other variables, such as employment status, sex, family size and age, did not show statistically significance at the 5 percent level. The model recommended by the authors is shown in Table 3.2. Variables include average total travel time, fraction of time spent in congested conditions, income of respondent, and the monetary cost of the trip. Their estimation results for each parameter associated with VOT estimation are presented in Table 3.3.

The three models (Cherchi and Ortuzar, 2003; Amador et al., 2004; and Brownstone et al., 2003) take into account a single variable of car/licenses, sex, and trip purpose, respectively. Using the RP data in Cagliari, Italia, Cherchi and Ortuzar (2003) estimated several nested logit models and mixed logit model, with not only linear but also non-linear specifications to test the effect of including socio-economic variables in the utility function in a modal choice (i.e., car, bus and train users). They found that the VOT estimated with a non-linear specification is strongly influenced by some socio-economic variables of the individual. From their results, they also found that a mixed logit model performs better than nested logit model with interaction terms. The second row in Table 3.2 presents the estimated parameters from the mixed logit model that they concluded the best model for VOT estimation. Note that the only significant personal attribute variable was car/licenses (the ratio between the number of cars in a family and the number of driving licenses). They found that all the other socio-economic variables improve the model but not strongly. It is somewhat surprising results, because no other models include this variable. It seems proper to interpret this result as a local context.

Table 3.1 Authors and Description of Variables

Author	Description of variables
(1) Small et al. (1999)	T_c = time spent in congested conditions, M = monetary cost, Y = household income (in thousand dollars)
(2) Cherchi and Ortuzar (2003)	$Car / Licences$: the ratio between the number of cars in a family and the number of driving licenses
(3) Amador et al. (2004)	Sex : dummy variable, 1 for men and 0 otherwise
(4) Brownstone et al. (2003)	*T*: travel time saving, *V*: variability (measured as the difference between median and 90th percentile time saving), *C*: toll, P : trip purpose dummy (1: commute trip, including trips to work or school, 2: non-commute)
(5) Algers et al. (1998)	*Age*: a dummy variable (1 for aged 45 or older, 0 otherwise), *Work*: a dummy variable (1 for a work trip, 0 otherwise), *Paid*: a dummy variable (1 if a person is gainfully employed, 0 otherwise), *Hhinc*: a variable indicating to which out of 14 household income classes the respondent belongs
(6) Small et al. (2003)	C : toll difference between express lane and regular lane I_m : dummy variable for medium household income ($60,000-$100,000) I_m : dummy variable for high household income (> $100,000) D : trip distance (10 miles) L : long commute dummy (> 45min) T : median travel time difference between two lanes R : unreliability of travel time (min) X_i : socio-economic variables (female dummy, age between 30 and 50 dummy, flexible arrival time dummy, household size in number of people)
(7) Konig et al. (2003)	ΔT : travel time difference, ΔC : travel cost difference
(8) Ghosh (2000)	T : travel time savings between the two routes (toll or non-toll), C : amount of toll ($)
(9) Whelan and Bates (2001)	$Cong$: congestion, (ratio of reported time spent in congested conditions to reported total travel time) $TwoAdult$: vehicle occupancy (two adults) $ThreeAdult$: vehicle occupancy (three or more adults) $NChild$: number of child in passenger $Visit1$, $Visit2$, $Visit3$, $Visit4$: trip sub-purpose, indicating visiting branch office, visiting client, attending business meeting, and attending seminar, respectively. $Age45$, $Age3544$: age of more than 45, and between 35-44, respectively $Female$: a dummy variable $Child$: a dummy variable, indicating the presence of children in the household $Passen$: a dummy variable, indicating that a respondent is a passenger $Fixarr$: fixed arrival time Inc : Income (£1,000) $Reim1$: reimburse now, indicating versions of the questionnaire that did not mention who would pay any additional costs $Reim2$: reimburse fixed, indicating to questionnaires in which the respondents were told that they would receive a fixed amount of reimbursement. $Selfemp$, $Parttime$, retired : occupation, indicating self-employed and part time workers, respectively.

Table 3.2 Various Utility Forms of VOT Estimation with Non-linear Function

	Utility Function	Value of Time (VOT)
(1)	$U=(\beta_t+\beta_Y(Y-65))T+\beta_c C+\beta_{fc}(T_c/T)+\varepsilon_i$	$\frac{\beta_t+\beta_Y(Y-65)}{\beta_c}$
(2)	$U=(\beta_t+\beta_{CL}Car/Licences)T+\beta_c C+\beta_i X_i+\varepsilon_i$	$\frac{\beta_t+\beta_{CL}Car/Licences}{\beta_c}$
(3)	$U=(\beta_t+\beta_{sex}Sex)T+\beta_c C+\beta_i X_i+\varepsilon_i$	$\frac{\beta_t+\beta_{sex}Sex}{\beta_c}$
(4)	$U=(\beta_{p1}P_1+\beta_{p2}P_2)T+(\beta_c+\beta_v V)C+\beta_i X_i+\varepsilon_i$	$\frac{\beta_{p1}P_1+\beta_{p2}P_2}{(\beta_c+\beta_v V)}$
(5)	$U=(\beta_t+\beta_{Tage}Age+\beta_{Twork}Work+\beta_{Tpaid}Paid+\beta_{Thhinc}Hhinc)T+(\beta_c+\beta_{Cwork}Work)C+\varepsilon_i$	$\frac{\beta_t+\beta_{Tage}Age+\beta_{Twork}Work+\beta_{Tpaid}Paid+\beta_{Thhinc}Hhinc}{(\beta_c+\beta_{Cwork}Work)}$
(6)	Model 1 (binary logit model, RP only): $U=(\beta_{d1}D+\beta_{d2}D^2+\beta_{d3}D^3)\Delta T+(\beta_c+\beta_m I_m+\beta_h I_h)\Delta C+\beta_r R+\beta_i X_i+\varepsilon_i$ Model 2 (mixed logit model, SP only): $U=(\beta_{lc1}L+\beta_{lc2}(1-L))\Delta T+(\beta_c+\beta_m I_m+\beta_h I_h)\Delta C+\beta_r R+\beta_i X_i+\varepsilon_i$	M1: $\frac{\beta_{d1}D+\beta_{d2}D^2+\beta_{d3}D^3}{(\beta_c+\beta_m I_m+\beta_h I_h)}$ M2: $\frac{\beta_{lc1}L+\beta_{lc2}(1-L)}{(\beta_c+\beta_m I_m+\beta_h I_h)}$
(7)	$U=(\beta_{t1}+\beta_{t2}\Delta T)\Delta T+(\beta_{c1}+\beta_{c2}\Delta C)\Delta C+\varepsilon_i$	$\frac{\beta_{t1}+2^*\beta_{t2}\Delta T}{(\beta_{c1}+2^*\beta_{c2}\Delta C)}$
(8)	$U=(\beta_t+\beta_{t2}T)T+(\beta_c+\beta_{c2}C)C+\beta_i X_i+\varepsilon_i$	$\frac{\beta_t+2^*\beta_{t2}T}{(\beta_c+\beta_{c2}C)}$
(9)	For business trip: $U=(\beta_t+\beta_{Tcong}Cong+\beta_{T2adult}TwoAdult+\beta_{T3adult}ThreeAdult+\beta_{Tnchild}NChild+\beta_{Tvisit1}Visit1+\beta_{Tvisit2}Visit2$ $+\beta_{Tvisit3}Visit3+\beta_{Tvisit4}Visit4+\beta_{TAge45}Age45+\beta_{Tfemale}Female+\beta_{TChild}Child+\beta_{Tpass}Passen+\beta_{Tfix}Fixarr)T$ $+(\beta_c+\beta_{Cinc}Inc+\beta_{Creim1}Reim1+\beta_{Creim2}Reim2)C+\varepsilon_i$ For commuting trip: $U=(\beta_t+\beta_{Tcong}Cong+\beta_{T1adult}OneAdult+\beta_{T2adult}TwoAdult+\beta_{T3adult}ThreeAdult+\beta_{Tnchild}NChild$ $+\beta_{Tselfemp}Selfemp+\beta_{Tpart}Parttime+\beta_{TAge}Age+\beta_{Tfemale}Female+\beta_{Tpass}Passen)T$ $+(\beta_c+\beta_{Cinc}Inc+\beta_{Creim1}Reim1+\beta_{Creim2}Reim2)C+\varepsilon_i$ For other trips: $U=(\beta_t+\beta_{T1adult}OneAdult+\beta_{T2adult}TwoAdult+\beta_{T3adult}ThreeAdult+\beta_{Tnchild}NChild+\beta_{Tretired}Retired$ $+\beta_{TAge3544}Age3544+\beta_{Tfemale}Female+\beta_{Tfix}Fixarr)T+(\beta_c+\beta_{Cinc}Inc+\beta_{Creim1}Reim1)C+\varepsilon_i$	$VOT=\frac{\partial U_i/\partial T_i}{\partial U_i/\partial C_i}$

Table 3.3 Results of Parameters Estimation

	Parameters estimation
(1)	β_T = -0.093, β_Y = -0.001, β_C = -0.926
(2)	β_t = -0.296, β_{CL} = 0.414, β_c = -0.056
(3)	β_t = -0.058, β_{sex} = 0.026, β_c = -0.003
(4)	β_{p1} = 0.193, β_{p2} = 0.423, β_C = -0.647, β_v = 0.135
(5)	β_T = -0.151, β_{Tage} = 0.065, β_{Twork} = -0.153, β_{Tpaid} = -0.056, β_{Thhinc} = -0.009, β_C = -0.102, β_{Cwork} = -0.327
(6)	M1: β_{d1} = -0.262, β_{d2} = 0.041, β_{d2} = -0.002, β_C = -1.344, β_m = 0.469, β_h = 0.905 M2: β_{lc1} = -0.289, β_{lc1} = -0.302, β_C = -1.547, β_m = -0.303, β_h = 0.163
(7)	β_{t1} = -0.206, β_{t2} = 3.572, β_{c1} = -0.288, β_{c2} = 1.319
(8)	RP data from FasTrak users: β_T = -0.770, β_{T2} = 0.038, β_C = -1.289, β_{C2} = 0.209 SP data from FasTrak users: β_T = -0.261, β_{T2} = 0.003, β_C = -0.908, β_{C2} = 0.010 SP data from other I-15 users: β_T = -0.330, β_{T2} = 0.006, β_C = -1.773, β_{C2} = 0.168 SP data from I-8 users: β_T = -0.179, β_{T2} = 0.003, β_C = -0.925, β_{C2} = 0.063
(9)	For business trip: β_T = -0.072, β_{Tcong} = -0.063, $\beta_{T2adult}$ = -0.039, $\beta_{T3adult}$ = 0.058, $\beta_{Tnchild}$ = -0.030, $\beta_{Tvisit1}$ = -0.021, $\beta_{Tvisit2}$ = -0.017, $\beta_{Tvisit3}$ = -0.012, $\beta_{Tvisit4}$ = -0.020, β_{TAge45} = 0.023, $\beta_{Tfemale}$ = -0.014, β_{TChild} = -0.005, β_{Tpass} = 0.016, β_{Tfix} = -0.011, β_C = -0.015, β_{CInc} = 0.0001, β_{Creim1} = 0.003, β_{Creim2} = 0.002 For commuting trip: β_T = -0.112, β_{Tcong} = -0.039, $\beta_{T1adult}$ = -0.039, $\beta_{T2adult}$ = 0.016, $\beta_{T3adult}$ = -0.158, $\beta_{Tnchild}$ = -0.039, $\beta_{Tselfemp}$ = -0.071, β_{Tpart} = 0.060, β_{TAge} = 0.002, $\beta_{Tfemale}$ = -0.020, β_{Tpass} = -0.042, β_C = -0.030, β_{CInc} = 0.0003, β_{Creim1} = 0.008, β_{Creim2} = 0.008 For other trips: β_T = -0.089, $\beta_{T1adult}$ = 0.026, $\beta_{T2adult}$ = 0.029, $\beta_{T3adult}$ = 0.060, $\beta_{Tnchild}$ = 0.011, $\beta_{Tretired}$ = -0.020, $\beta_{TAge3544}$ = -0.030, $\beta_{Tfemale}$ = 0.013, β_{Tfix} = -0.022, β_C = -0.021, β_{CInc} = 0.0001, β_{Creim1} = 0.001

Amador et al. (2004) explored several model specifications to find out the presence of preference heterogeneity, based on the variables of sex, age, possession of a vehicle, and family income level, and etc. They used RP data from students of University of La Laguna, Spain for a mode choice. They found that benefit measures of travel time savings are sensitive to preference heterogeneity assumptions. Accordingly, they argued that VOT could be underestimated if the traditional assumption of homogeneous VOT is made. However, the dummy variable of sex was the only variable in their model that is significantly interacted with travel time. Using a cost-squared variable, they tested to see if there is an income effect, but the income variable was not significant. The best model using multinomial logit is shown on the third low of Table 3.2.

Using RP data from I-15 HOT lane value pricing project in San Diego, Brownstone et al. (2003) estimated commuters' willingness to pay to reduce commute time. In their mode choice model, each traveler has three alternatives: 1) solo driving on the main lanes, 2) solo driving using toll lane, and 3) driving with others in a carpool. Although they considered demographic/social attribute data, such as gender, age and education, into the random utility model, these variables did not affect the computation of VOT. Instead, they estimated willingness to pay, by accounting for the interaction between toll and variability of travel time savings, which is measured as the difference between median and 90th percentile time saving. From their choice model, they found that those more likely to pay to use the HOT lanes are commuters, who have high household income ($100,000 or more), are women, are between the ages of 35 and 45, have higher education, and are homeowners.

Algers et al. (1998) estimated VOT for long-distance car travel in Sweden, using mixed logit model that allows several parameters to vary in the population. The main conclusion was that the estimated VOT is very sensitive to how the model is specified. It was found that VOT is significantly lower when the coefficients are assumed to be normally distributed in the population than the traditional fixed coefficients assumption. The authors developed non-linear functional form of indirect utility function, accounting for squares of the cost and time variables, as well as cost and time interacted with socio-economic variables. As results of their estimations, they suggest the model shown on Table 3.2, including the following socio-economic variables: age, work trip purpose, gainfully employed, and household income class.

Small et al. (2003) analyzed the travelers' preference by combining revealed and stated commuter choices in the context where a driver can choose a toll lane or regular congested free lanes. In the utility model, they have taken into account interactions of household income to the cost difference, and distance traveled (including squared and cubed form) and long commute dummy to the travel time difference. Although they considered other socio-economic variables in the random utility model, they conclude that those socio-economic variables did not affect the decision of VOT, since they did not interact with cost and time variables. The authors have concluded that there are substantial heterogeneity in travelers' value of time and value of reliability.

The utility models proposed by Konig et al. (2003) and Ghosh (2003) illustrates the second-order terms between socio-economic variables and LOS variables in the non-linear systematic function. Konig et al. (2003) showed the preliminary result of study in Switzerland measuring VOT by mode and trip purpose, using SP data. To incorporate non-linear specification, they formulated the utility function using differences of travel time and costs, their squared values and interactions. The estimated coefficients are shown on Table 3.3. On the other hand, Ghosh (2000) estimated VOT using RP and SP data from I-15 Value Pricing project in San Diego, California. The estimated results are shown in Table 3.3. The VOT used in these models is a non-linear function of toll and travel time savings between the two routes (toll vs. non-toll roads).

The Working Paper of Whelan and Bates (2001) present a full research on differences in the value of time by market segment. It is based on a utility model to estimate VOT for each trip purpose (business, commuting and other). In the research, they show that how the base model is affected by each market segment. The market segment includes income (£1,000), trip distance, cost reimbursement, congestion, vehicle occupancy, trip sub-purpose, occupation, age group, gender, household type, free time, respondent type, time constraints and geographical region. From each segmentation analysis, they developed a final set of models that allow VOT to vary across a range of market segments. The final segmentation models are shown in the final row of Table 3.2.

3.3 Literature Review on Departure Time Choice Modeling

Over the past decades the departure time choice models are well studied due to the importance of dealing with peak-period traffic congestion. Table 3.4 summarizes selected departure time choice models in the U.S. The dominant model of departure time choice in practice and academic research is a discrete choice model (i.e., multinomial logit model). Small (1982) developed a multinomial logit choice model to model activity scheduling at the individual level using demographic variables. The utility function includes good indicators of departure time choice (i.e., travel time, schedule delay, household structure, carpool, arrival flexibility, occupation type). However, the model is work trip only. In addition, the choice sets (5-min time intervals) are based on arrival time at work, rather than departure time from home. Cambridge Systematics (2005) developed a time of day model for both trip-based and tour-based models (Abou Zeid, et al., 2006). Although their trip based model considers both transportation attributes and individual attributes, arrival times were modeled for trips from home, while departure times were modeled for trips to home and for non-home based trips.

Mahmassani and other investigators (Mahmassani and Chang, 1985; Srinivasan and Mahmassani, 1999; and Mahmassani and Liu, 1999) proposed a couple of departure time choice models based on the boundedly rational behavior rules. According to the boundedly rational principal, drivers are postulated to accept an alternative if it is both satisfactory and sufficient. A traveler would adjust his/her departure time whenever his/her schedule delay exceeds some tolerable value, referred to as individual "indifference band." The departure time switching model based on this concept was initially developed by Mahmassani and Chang (1985) and extended by Mahmassani and other investigators. For example, Mahmassani et al. (1990) developed a binomial logit model of departure time switching. The variables used in the utility model are travel time, lateness tolerance at work, preferred arrival time, job type, and use of information. However, their model is limited to work trip only. Mahmassani and Liu (1999) analyzed day-to-day dynamics of commuters' departure time and route switching in response to Advanced Traveler Information Systems (ATIS), using a multinomial probit model.

Table 3.4 Summary of Selected Departure Time Choice Models in the U.S.

Paper	Work/ Non-work	Time Period	Methodology	Variables
Abkowitz (1981)	W	AM	Multinomial logit (5-min. intervals)	Arrival flexibility, mode, occupation, location, income, age, travel time
Burris & Pendyala (2002)	Both	Full day	Binary logit (changing DT or not)	Flexibility in time of travel, retirement status (Age), flextime availability, income, trip purpose,
Caplice & Mahmassani (1992)	W	AM/PM	Multinomial logit (1) do not switch DT or route, (2) switch DT only, (3) switch route only, & (4) switch both DT and route	Travel time, lateness tolerance at work, preferred arrival time, abundance of alternative routes, home ownership (rent or not), gender, use of information
Hendrickson & Plank (1984)	W	AM	Multinomial logit – joint mode & departure time (10-min. intervals)	Free flow travel time, congestion time, cost/income, time early, time late, access time, wait time
Levinson & Kumar (1993)	Both	PM	Binomial logit (peak & shoulder)	Congested & free-flow travel time, distance
Mahmassani, Caplice & Walton (1990)	W	AM/PM	Binary logit (switching DT or not)	Travel time, lateness tolerance at work, preferred arrival time, job type, use of information
Mannering (1989)	W	AM	Poisson regression (# departure time changes/month)	Travel time, work time flexibility, age, marital status
Small (1992)	W	AM	Multinomial logit (5-min arrival time intervals)	Travel time, schedule delay, household structure, carpool, arrival flexibility, occupation type

Other types of models include departure time choice decisions under congestion pricing (Yamamoto et al., 2000; Burris and Pendyala, 2002; Saleh and Farrell, 2005), joint models of departure time choice combined with mode and/or route choice (Hendrickson and Plank, 1984; Mannering, 1989; Yamamoto et al., 2000), continuous models (Wang, 1996; Bhat and Steed, 2002), and activity rescheduling models (Ettema and Timmermans, 2003; Joh et al., 2005). For example, Burris and Pendyala (2002) developed a binomial logit model to examine whether a traveler would change time of travel in response to variable toll pricing and the frequency of change. The variables include trip purpose, flexibility in time of travel, retirement status (this is corresponding to age), flextime availability at work, and household income (> $75,000). Burris and Pendyala (2002) estimated the model, based on travel survey data (RP survey) collected in variable toll pricing system in the Lee County, Florida. The analysis showed that retired travelers with low income and flexible schedule are more likely to change their time of travel in response to variable tolls. Hendrickson and Plank (1984) modeled mode and departure time choices simultaneously. They examined the choice sensitivity of individuals, and showed the effect of new policies such as tolls, exclusive bus lanes, or changes in congestion. Their results suggested that the departure time decision were more elastic than the choice of mode. Continuous models of departure time choice were proposed by, for example, Wang (1996) and Bhat and Steed (2002). Bhat and Steed (2002) proposed a continuous time model to analyze departure time choice for urban shopping trip, using activity survey data collected in the Dallas-Fort Worth metropolitan area. They found that there are significant effects of individual and household socio-demographics, employment attributes and trip-related characteristics on departure time choice. Some studies investigated how activities are rescheduled. Ettema and Timmermans (2003) developed a departure time choice model by incorporating activity timing and duration. They presented interrelationship between departure times of different trips. However, their model estimation was based on the assumption of constant travel times due to lack of suitable data on travel time variability by time of day. Joh et al. (2005) analyzed activity scheduling and rescheduling behavior using such variables as household and personal characteristics, activity flexibility, fixed activity attributes, contextual activity characteristics, and schedule/tour characteristics. The analysis showed the frequency of occurrence of modifications of planned activity schedule. As results, Joh et al. (2005) found that activity timing and duration were much more frequently modified than activity location and co-scheduling with peers.

Chapter 4. Value of Time and Dynamic Value Pricing

The objectives of this research are to utilize non-linear, individual VOT function in route choice of a HOT lane value pricing system and to find out 15-min dynamic toll rates that vary by level of service in the HOT lane. A volume/capacity (V/C) ratio is used as an indicator of level of service. To achieve these objectives, a microscopic simulation model referred to as TRANSIMS was selected. The functions of TRANSIMS were improved by modifying the Router and the associated source codes. TRANSIMS has unique features to analyze dynamic HOT lane value pricing in a large transportation network by taking into account an individual-based VOT function. TRANSIMS differs from current travel demand forecasting methods in its underlying concepts and structure. These differences include a consistent and continuous representation of time; a detailed representation of persons and households; time-dependent routing; and a person-based microsimulator. TRANSIMS Microsimulator is the only simulation tool that maintains the identity of the traveler throughout the simulation, and is capable of accessing the database of each individual (e.g., income, age, trip purpose, etc.). In other words, it traces the movement of people as well as vehicles on a second-by-second basis. In addition, TRANSIMS route planner utilizes a time-dependent, individually-based route choice model that is suitable for considering each individual VOT.

The enhancements to TRANSIMS were made in the two different versions of TRANSIMS. The old version (v.3.0.1), which was available only in the Linux system, was first modified and tested on a simplified network. However, the TRANSIMS environment has significantly changed over the past few years. TRANSIMS is now open to the public. The software, source codes, tutorials, documents, and case studies are available throughout the TRANSIMS Open Source website (2008). New TRANSIMS software (since version 4.0.1) is now available for both Windows and Linux machines. FHWA and open source community members continue working to improve TRANSIMS codes, methods and documentation. However, there are a few issues that still need to be improved. For example, even though new TRANSIMS allows toll application (i.e., constant or time-varying tolls), the Router takes only a constant conversion factor from a cost value to a travel time value, thus does not accept various VOT or a distribution of VOT over the population. Accordingly, the new version (v.4.0.1) was also modified and tested as a part of the FHWA research project. The modified TRANSIMS was tested using two different data sets, including 1) a simplified network, and 2) a real-world network using data from Portland, Oregon. Note that some of discussions and analysis in this chapter were already published in the journal paper by the author (Lee and Hobeika, 2007).

This chapter presents a non-linear, individual VOT function, and discusses the proposed generalized travel time function. The proposed simulation methodology and specific modifications to enhance to TRANSIMS are also discussed. Finally, the two experimental analyses are presented.

4.1 Non-linear, Individual Value of Time and Generalized Travel Time Function

Using constant VOT is not suitable for the analysis of the HOT lane value pricing, as discussed in the previous chapter. The following two sections introduce an individual, non-linear VOT function that used for this research, and presents a generalized travel time function that simultaneously taking into account individual VOT and dynamic tolls.

4.1.1 Non-linear, individual value of time (VOT)

Heterogeneous VOT for each traveler is being utilized in this research. The assumptions made in this research are: 1) an individual driver has a different VOT from other drivers, 2) a VOT function consists of known and measurable components, and 3) VOT is a non-linear function based on each driver's socio-economic and travel characteristics. A number of models have been suggested to estimate a non-linear function of VOT for each individual driver based on measurable characteristics of socio-economic variables, and they are discussed in the literature reviews (Chapter 3). Among the models, this research adopts and modifies, as a default VOT function, the model originally proposed by Algers et al. (1998), as shown in Eq. 4.1. This model was adopted because it includes more variables such as socio-economic factors than any other models. The model shows plausible relationships in estimating each individual's VOT. Most importantly, all variables are also available as TRANSIMS input data.

$$VOT_i = \frac{(\beta_T + \beta_{Tage} Age_i + \beta_{Twork} Work_i + \beta_{TEmp} Employee + \beta_{Thhinc} Hhinc_i)}{(\beta_C + \beta_{Cwork} Work_i)} \tag{4.1}$$

$$= \frac{(0.8 - 0.7 Age_i + Work_i + 0.1 Employee_i + 0.2 Hhinc_i)}{(0.06 + 0.02 Work_i)} \tag{4.2}$$

where,

Age: A dummy variable (1 for aged 45 or older, 0 otherwise),

$Work$: A dummy variable (1 for a work trip, 0 otherwise),

$Employee$: A dummy variable (1 if a person is full-time employee, 0 otherwise),

$Hhinc$: An income variable

The modified VOT function for an individual i is shown in Eq. 4.2, taking into account three socio-economic variables (age, employment status and household income class) and one trip characteristic (trip purpose). Note that all coefficients in Eq. 4.2 show reasonable signs. The variable $Work$ is positively interacted with travel time as well as with travel cost. The variable Age is negatively interacted with travel time only, showing that travelers aged 45 or more seem less sensitive to travel time than do younger travelers. The income variable $Hhinc_i$ is positively related to VOT and is divided into 7 household income groups (e.g., income group 1 represents the lowest income group). Different values are applied for each income group depending on trip purpose (for work-trip, 1.00, 4.33, 7.66, 11.00, 19.00, 27.00, and 35.00 for income group 1, 2, 3, 4, 5, 6 and 7 respectively; for non-work trip, 1.00, 1.17, 1.33, 1.50, 1.67, 1.83 and 2.00 for income group 1, 2, 3, 4, 5, 6 and 7 respectively).

It should be noted that the equation shows the modified coefficients of each variable, since the Algers' model is based on Swedish values. The modification was conducted to reflect the real data of I-15 HOT lane value pricing system as closely as possible. For example, Steimetz and Brownstone (2005) estimated the heterogeneous VOT using the panel survey in I-15. Table 4.1 presents VOT results studied by Steimetz and Brownstone and compares to the calculated VOTs using the proposed VOT equation (Eq.

4.2). When using the modified VOT function (Eq. 4.2), the lowest VOT is 5.00 (\$/hr) for a person, who is older than 45 years, has a non-work trip, is non-employed, and belongs to the lowest income group, whereas the highest VOT is 111.25 (\$/hr) for a person, who is younger than 45 years, has a work trip, is employed, and belongs to the highest income group. The ranges of calculated VOT using the modified VOT function are consistent with the estimates obtained by Steimetz and Brownstone (2005).

It should be also noted that the VOT function (Eq. 4.2) is used as a default model. The modified TRANSIMS (v.4.0.1) was developed to accept different VOT coefficients, so that it allows for a user to specify his/her own coefficients for VOT in the VOT coefficients file. This file improves the flexibility for the user in using different VOT functions. Note that the intent of this research is not to develop an accurate VOT model. The issues surrounding the model estimation/verification are beyond the scope of this research.

Table 4.1 Comparison of VOTs Results Studied by Steimetz and Brownstone (2005) with VOTs Calculated Using Proposed Model

		Steimetz and Brownstone's results		VOT ranges from our modified model ($/hr) and characteristics of a person
		VOT median ($/hr)	25-percentile and 75-percentile	
Work trips	Income > $80K	64.90	41.48, 111.78	41.25 – 101.25 (work trip, non-employee, age ≥ 45, income group 3 to 5) 42.50 – 102.50 (work trip, employee, age ≥ 45, income group 3 to 5) 50.00 – 110.00 (work trip, non-employee, age<45, income group 3 to 5) 51.25 – 111.25 (work trip, employee, age<45, income group 3 to 5)
	Income < $80K	21.52	16.21, 28.79	16.25 – 26.25 (work trip, non-employee, age ≥ 45, income group 1 to 2) 17.50 – 27.50 (work trip, employee, age ≥ 45, income group 1 to 2) 25.00 – 35.00 (work trip, non-employee, age<45, income group 1 to 2) 26.25 – 36.25 (work trip, employee, age<45, income group 1 to 2)
	Full-time workers	44.12	25.81, 70.36	17.50 – 70.00 (work trip, employee, age ≥ 45, income group 1 to 4) 26.25 – 78.75 (work trip, employee, age<45, income group 1 to 4)
	Part-time workers	15.65	11.58, 21.50	16.25 – 26.25 (work trip, non-employee, age ≥ 45, income group 1 to 2) 25.00 – 35.00 (work trip, non-employee, age<45, income group 1 to 2)
Non-work trips	Income > $80K	14.35	10.37, 21.35	6.67 – 8.33 (non-work trip, non-employee, age ≥ 45, income group 3 to 5) 8.33 – 10.00 (non-work trip, employee, age ≥ 45, income group 3 to 5) 18.33 – 20.00 (non-work trip, non-employee, age<45, income group 3 to 5) 20.00 – 21.667 (non-work trip, employee, age<45, income group 3 to 5)
	Income < $80K	9.60	7.16, 12.92	5.00 – 5.83 (non-work trip, non-employee, age ≥ 45, income group 1 to 2) 6.67 – 7.50 (non-work trip, employee, age ≥ 45, income group 1 to 2) 16.67 – 17.50 (non-work trip, non-employee, age<45, income group 1 to 2) 18.33 – 19.17 (non-work trip, employee, age<45, income group 1 to 2)
	Full-time workers	10.83	7.97, 14.43	6.67 – 10.00 (non-work trip, employee, age ≥ 45, income group 1 to 5)
	Part-time workers	7.25	5.53, 9.57	5.00 – 8.33 (non-work trip, non-employee, age ≥ 45, income group 1 to 5)

4.1.2 Generalized travel time function

The old (3.0.1) and new (4.0.1) versions of the TRANSIMS do not support a generalized travel time (or cost) function applicable to the HOT lane value pricing system that can simultaneously take into account individual VOT and dynamic tolls. The existing Router is only based on link travel times. In order to find the shortest path, the existing Router takes free-flow travel times in the first run or travel times obtained from the Microsimulator for the rest of runs. The modified Router in this research searches the shortest path for each traveler based on a generalized travel time function. The form of the generalized travel time function is shown in Eq. 4.3. It consists of two components. The first component is the 15-min time-dependent travel time obtained from the Microsimulator. The second component is the additional travel time by taking account of ratio of toll to VOT to convert monetary value (toll rate, \$) to time value (seconds). The vehicle characteristic variable (α) is a dummy variable, having 0 for HOVs or 1 for SOVs. Since HOV travelers can travel on any route, including HOT lanes, without paying a toll, they do not have an additional travel times on the second term in Eq. 4.3.

$$GTT_i(t) = TT(t) + \alpha \left[\frac{toll(t)}{VOT_i} \right] \tag{4.3}$$

where,

$GTT_i(t)$: The generalized perceived travel time for traveler i at time t

α : A dummy variable (1 for SOV, otherwise 0)

$toll(t)$: Toll rate (cents) at time t

VOT_i: Value of time for traveler i (cents/second converted from \$/hr format)

$TT(t)$: Link travel times obtained from the Microsimulator

4.2 Simulation Methodology

In the feedback process of TRANSIMS, the Microsimulator feeds back the experienced travel times to the Router to search for the time-dependent shortest travel time paths. The overall structure and processes of the proposed approach to evaluate dynamic value pricing are depicted in Figure 4.1, and are summarized below.

1) Initialize by setting *TRAV*= *POP*= {all travelers}, and set the system-based objective value, *Y**, to be infinity. Tolls are set at zero initially. Run the first simulation time slot to determine the 15-min dynamic toll for this time slot.

2) Select an individual traveler α, and calculate his/her VOT. Based on the "current" toll rate and VOT, apply the generalized travel time function to find the shortest path for the traveler. Let the Routed Travel Time (RT_α) for each traveler α be as that obtained from the Router.

3) Simulate the travelers' patterns using the Microsimulator, compute the Experienced Travel Time (ET_α) for all travelers α in *POP*, and find the link volumes and link travel times.

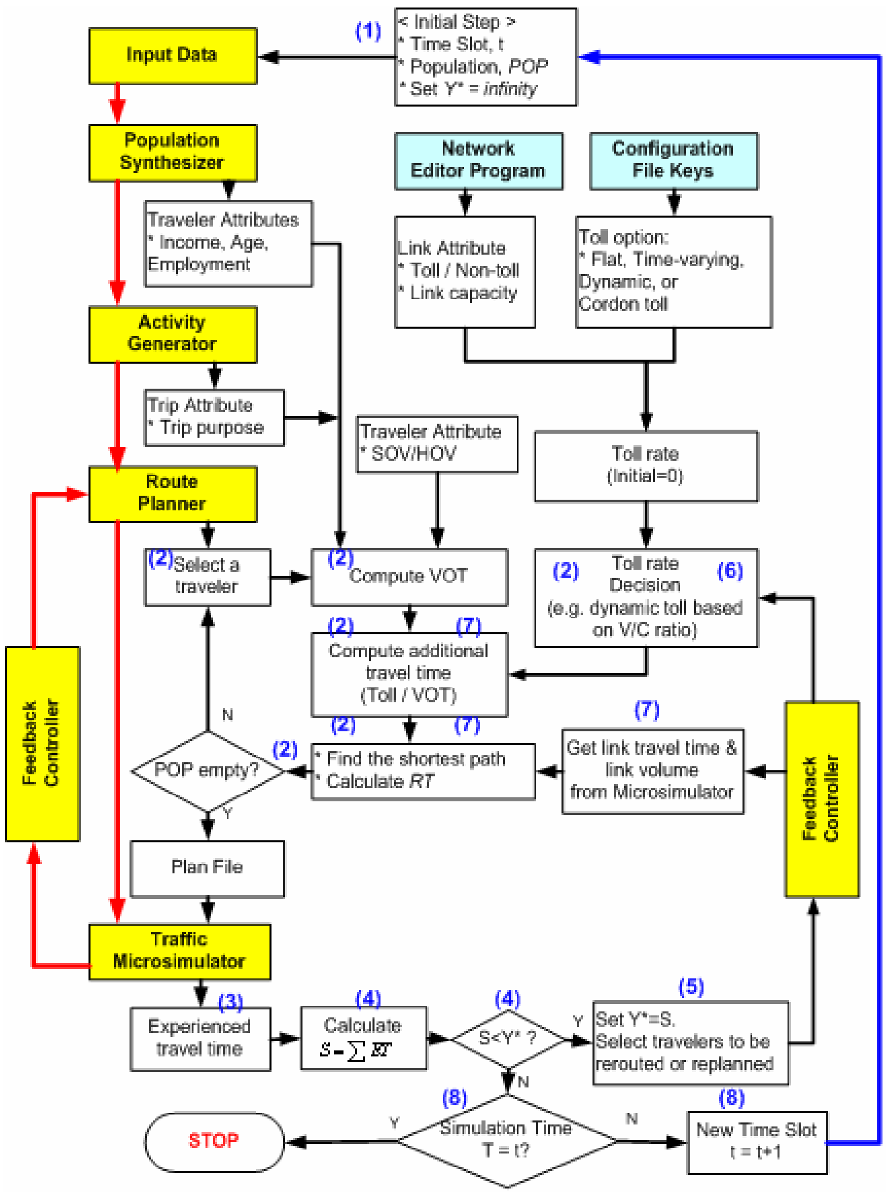
4) Calculate $S = \sum_\alpha ET_\alpha$. If $S < Y^*$ go to Step 5; otherwise, go to Step 8.

5) Set $Y^* = S$ and select travelers who have the highest ET_α / RT_α ratio among travelers having $ET_\alpha / RT_\alpha > 1.1$. Set *TRAV*={selected travelers}. If *TRAV* is empty, go to Step 8; otherwise, go to Step 6.

6) The Router reads link volume on the HOT lane, compute V/C ratio, reads a base toll from the look-up table, and updates dynamic toll for this time slot.

7) Find the shortest paths (via the Router) for travelers in *TRAV* using the generalized travel times. Let RT_α denote the revised Routed Travel Time for each traveler $\alpha \in TRAV$. Go to Step 3.

8) If this time slot is the overall simulation time (T=t), then stop; otherwise, prepare input data for the next time slot, and go to Step 2.



* Note: the numbers in parentheses correspond to the steps discussed on the previous page.

Figure 4.1 Proposed simulation process.

4.2.1 Simulation time slot

As an initial step, overall simulation time horizon *T* is divided into *n* time slots. Partitioning time slots with 15-min time interval is required to find out a 15-min dynamic toll at each time slot. For instance, the first time slot (i.e., $\textit{TimeSlot}_{0:00\text{-}0:15}$) is run to find out the 15-min dynamic toll for the time period of 0:00-0:15 a.m. The simulation involves several runs throughout the Router/Microsimulator feedback loop until stopping criteria meet. Once the stabilized network is reached and the 15-min dynamic toll is decided for this time period (0:00-0:15 a.m.), the second time slot ($\textit{TimeSlot}_{0:15\text{-}0:30}$) is prepared and run to find the 15-min dynamic toll for the time period of 0:15-0:30 a.m. This process continues until overall simulation time horizon is simulated. Although this process (i.e., the first time slot simulates to find the 15-min dynamic toll for the time period of 0:00-0:15 a.m.) is ideal, it would not be an efficient way because not much traffic are expected to travel on the network in the early morning. Accordingly, a reasonable assumption can be made, for instance, by assuming that there are no tolls until 6 a.m. from the midnight. In this case, the first time slot would be a simulation run for the time period of 0:00-6:00 a.m. without any toll. The second time slot is a simulation run to find out the 15-min dynamic toll for the time period of 6:00-6:15 a.m.

4.2.2 Traveler and trip characteristics for VOT calculation

The Router computes VOT for each traveler using Eq. 4.2. The traveler's ID, household ID and activity ID numbers are used to match and extract appropriate data. The traveler attributes of socio-economic data are extracted from the output of the Population Synthesizer module. The trip attribute of trip purpose is extracted from the output of the Activity Generator module. The traveler attribute whether the vehicle is SOV or HOV is also extracted from the activity file. The calculated VOT is incorporated into the generalized travel time function.

4.2.3 Route Planner

The role of the Router is significant in this study as it carries the following functions: 1) to compute VOT based on each traveler and the corresponding trip characteristics, 2) to obtain link travel times from the Microsimulator, 3) to compute and/or update dynamic toll rates, and 4) to find the shortest path for each traveler based on the generalized travel times (Eq. 4.3). The Router finds the path of each trip conducted by a traveler at a time. It finds the shortest path using the modified Dijkstra's algorithm. The fundamental logic of the route choice behavior in TRANSIMS is that a traveler chooses the route to minimize his/her utility (travel time or travel costs if available). To support the value pricing system in TRANSIMS, the Router should simultaneously accept individual VOT and dynamically varying tolls in addition to link travel times.

4.2.4 Traffic Microsimulator, feedback process and stabilization

Based on the plans for each traveler generated by the Router, the Microsimulator simulates each vehicle second-by-second basis. After the Microsimulator, a subset of travelers is selected to be re-routed. The selection is based on the "travel time difference method." The Routed Travel Time (*RT*) obtained by the Router output is compared with the Experienced Travel Time (*ET*) computed by the Microsimulator output to determine if a trip is eligible for re-routing. If the *ET* is greater than 10% of the *RT*, the trip is eligible for re-routing. The feedback loop is conducted between the Router and the Microsimulator until the stopping criteria meet. This study utilizes two stopping criteria to determine when to stop the simulation for each time slot.

1. Minimum number of iteration (5 or 10 iterations)

The minimum number of iterations for each time slot is set as 5. However, once we obtain 15-min dynamic toll rates for all simulation time slots, the minimum number of iterations for the final simulation time slot is set as 10 to make sure that the stabilized network is reached.

2. Summation of the experienced travel times (*ET*) over all travelers

Another stopping criterion for each simulation time slot is the smallest summation of *ET* over all travelers referred to as *Y** (Jeihani et al., 2006). This value (*Y**) is initially set as an infinite number (i.e., huge number), and then it is compared with *S*, which is a variable equal to the summation of the *ET* of all travelers. If *S* is smaller than *Y**, then *Y** is replaced by *S* and the feedback loop between the Router and the Microsimulator continues.

Otherwise, the feedback loop stops, because the summation of the *ET* over all travelers does not strictly decrease. In other words, if there is no decrease in total travel times for all travelers, then the program stops, and convergence is declared. For example, Figure 4.2 shows the summation of *ET* over all travelers in the case of the simulation time slot #6 (this is equivalent to find the 15-min dynamic toll rate from 7:00 a.m. to 7:15 a.m.) in the case study #2 that simulates using data from Portland, OR. After the 5th iteration, which is the minimum number of simulation, the summation of *ET* is 41,099 hrs and it is decreased until the 10th iteration (36,406 hrs). However, the summation of *ET* is increased to 37,003 hrs in the 11th iteration. Accordingly, the simulation stops, and the results of the 10th iteration are obtained as the final simulation run for this time slot.

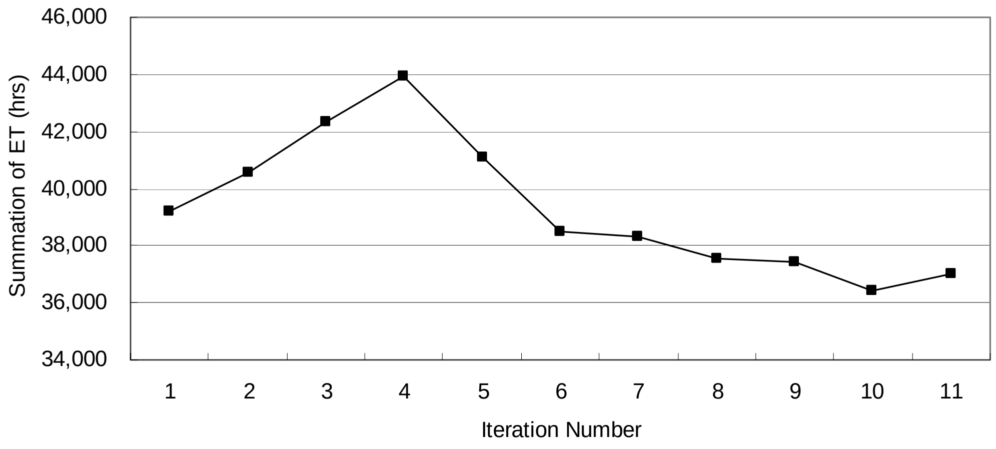


Figure 4.2 Summation of the experienced travel times over all travelers.

Once a stabilized network is reached throughout this loop and the dynamic toll is determined for this particular time slot, the next time slot is processed. This proposed heuristic approach requires several runs of TRANSIMS and takes significant simulation times. However, partitioning time slots with 15-min interval is necessary because it would be difficult to find out stabilized network and 15-min dynamic tolls simultaneously in one simulation process. The network condition and the dynamic tolls affect each other. The toll is updated by the traffic condition, at the same time the traffic condition will be changed in the next iteration according to the updated toll levels. Thus, this heuristic approach was proposed and implemented in this research to find a solution in terms of stabilized traffic volume and 15-min dynamic tolls.

4.3 Enhancements to TRANSIMS

To support the individual VOT and 15-min dynamic tolls, two different versions (v.3.0.1 and v.4.0.1) of TRANSIMS were modified. The functions of the Router are similar in both versions. The only difference is the input file to the Router. Appendix A summarizes added/modified source codes of the latest version of TRANSIMS, and Appendix B presents source codes. The following sections explain and focus on the enhancements to the latest version of TRANSIMS.

4.3.1 New control parameters

The several new control parameters were inserted to the source codes of the Router and associated codes (v.4.0.1) to support the individual VOT and 15-min dynamic tolls. They are shown in Table 4.2. A user defines these control parameters (i.e., a file name, location of the file, etc.) in a Router control file. An example of a VOT coefficient file is shown in Table 4.3.

Table 4.2 New Control Parameters

Parameter Name	Description and Usage
NET_DTOLL_TABLE	A file name and location of a dynamic toll file.
NET_LOOKUPTABLE_TABLE	A file name and location of a look up table for base toll.
VALUE_OF_TIME	This key (Yes or No) defines whether a VOT function is utilized in the Router. If it is Yes, then VOT is calculated in the Router, otherwise the Router does not take into account VOT.
VOTPOPULATION_FILE	A file name to read a population file for VOT calculation.
VOTHOUSEHOLD_FILE	A file name to read a household file for VOT calculation.
VOTCOEFFICIENTS_FILE	A file name of the VOT coefficient file to define different coefficients for the VOT function. This key is optional, and if not provided, the default VOT coefficients are used.
VALUE_OF_TIME_FILE_SOV	A file name for the VOT output file. This key is optional, and if provided, the Router generates an output VOT file, including individual socio-economic and trip data and the computed VOT.
TOLL_INCREMENT	This key defines how much a new toll increases. The default value is 50 cents.
TOLL_DECREMENT	This key defines how much a new toll decreases. The default value is 25 cents.

Table 4.3 VOT Coefficient File

Nume_Const	Denom_Const	Age	Work	Denom_Work	Employee	Income
0.8	0.06	-0.7	1	0.02	0.1	0.2

4.3.2 Dynamic toll update

For the application of 15-min dynamic tolls, the Router requires two input files, including a toll file and a look-up table file. The toll file consists of nine fields as shown in Table 4.4. The first column is the index number for a toll record number. The Key column is an identification key. There are two options for the user for this key; that is, 1 or -1. If it is 1, then toll for this time period will be updated. And, once it is updated, the Router converts this key to 0, so that there is no more update in this iteration and that it identifies whether the toll is updated or not. If the key is -1, this means that the Router accepts the current toll rate without any update, because the toll in this time period was already updated in the previous simulation time slots or it is not ready to update. Table 4.4 shows a key number 1 in the time period from 21,600 seconds from midnight (equivalent to 6:00 a.m.) to 22,500 seconds from midnight (equivalent to 6:15 a.m.). In other words, the Router will update a toll for this specific time period only.

Table 4.4 Toll File for Dynamic Toll Update

INDEX	KEY	LINK	NODE	START	END	USE	TOLL	NOTES
1	1	12103	70103	21600	22500	SOV	0	dynamictoll
2	-1	12103	70103	22500	23400	SOV	0	dynamictoll
3	-1	12103	70103	23400	24300	SOV	0	dynamictoll
4	-1	12103	70103	24300	25200	SOV	0	dynamictoll
5	-1	12103	70103	25200	26100	SOV	0	dynamictoll
6	-1	12103	70103	26100	27000	SOV	0	dynamictoll
7	-1	12103	70103	27000	27900	SOV	0	dynamictoll
8	-1	12103	70103	27900	28800	SOV	0	dynamictoll

The dynamic toll is updated every 15 minutes, based on the congestion level (V/C ratio) in the HOT lane. The volume is read from the link delay file (Microsimulator output) and the capacity is read from the link file. Then, the Router computes V/C ratio on the HOT lane. This ratio represents the traffic condition on the HOT lane and is used to read a toll rate from the look-up table that is pre-determined by the user. For example, Table 4.5 shows the lower and upper V/C ratios and the corresponding toll rate (cents).

The dynamic toll rate decision used in this study is similar to the San Diego's I-15 "FasTrak" system. In the I-15 HOT lane system, the traffic volume counted from loop detectors is compared to a look-up table to determine the price, which is pre-determined by San Diego Association of Governments. The dynamic tolls in the I-15 HOT lane are designed to change every 6 min in $0.25 increments. Similarly, the toll rates based on the V/C ratios are predefined for this study (Table 4.5). These toll rates are referred to as the base tolls, which will be compared to an old toll for update.

Table 4.5 Look-up Table

INDEX	LOWVC	HIGHVC	TOLL
1	0	0.55	0
2	0.55	0.6	50
3	0.6	0.65	100
4	0.65	0.7	150
5	0.7	0.75	200
6	0.75	0.8	250
7	0.8	0.85	300
8	0.85	0.9	350
9	0.9	2	400

Table 4.6 illustrates how a dynamic toll is updated for the time period from 6:00 a.m. to 6:15 a.m. throughout 6 iterations runs. In the first iteration, the Router reads the HOT lane link volume, which is obtained from the previous Microsimulator output. The Router computes V/C ratio and it is 0.545. The Router now opens the look-up table file and reads a base toll (0 cents). The old toll is the toll rate obtained from the previous iteration of the Router. A new toll is determined based on the following rules.

- If (base toll – old toll ≥ TOLL_INCREMENT)

  then, (new toll = old toll + TOLL_INCREMENT)
- Else if (base toll – old toll > – TOLL_DECREMENT)

  then, (new toll = old toll)
- Else

  (new toll = old toll – TOLL_DECREMENT)

In other words, the change of rates depends on the difference between the old toll and the base toll. A new toll rate is increased (or decreased) only if the difference is greater than or equal to the value of TOLL_INCREMENT (or, less than or equal to the negative value of TOLL_DECREMENT). It should be noted that these rules were designed by the investigator to avoid abrupt toll rate changes, since the output might oscillate if the amount of toll updates is too large.

Table 4.6 Example of How Dynamic Toll is Updated on Iteration-by-iteration

Iteration number	Old toll ($)	V/C ratio	Base toll ($)	Updated toll ($)	Summation of *ET* (hrs) from Msim
1	0	0.545	0	0	39,514
2	0	0.678	1.50	0.50	39,584
3	0.50	0.676	1.50	1.00	38,921
4	1.00	0.676	1.50	1.50	38,968
5	1.50	0.657	1.50	**1.50**	37,667
6	1.50	0.659	1.50	1.50	37,697

As shown in Table 4.6, the old toll of the first iteration is $0. Because there is no difference between the base toll and old toll, the toll is not updated. In the second iteration of the Router, the old toll is still $0. Based on the V/C ratio of 0.678, the base toll is now $1.50, which is greater than the old toll ($0). Because the difference of base toll from old toll greater than the amount of TOLL_INCREMENT ($0.50), the Router updates a new toll to $0.50, by increasing by $0.50. This toll rate ($0.50) now becomes the old toll in the third iteration of the Router. In the third iteration, the difference of base toll ($1.50) from the old toll ($0.50) is greater than the amount of TOLL_INCREMENT, thus the Router updates a new toll to $1.00, by increasing by $50 cents. In the 4th iteration, the Router updates a new toll to $1.50. However, tolls are not increased in the 5th and 6th iterations because the base toll and the old toll is the same. This iteration process continued until the 6th iteration. After the 6th iteration, the summation of *ET* of all travelers from the Microsimulator output did not decrease. Accordingly, the final 15-min dynamic toll for this time period (6:00-6:15 a.m.) is determined $1.50, which is the result of the 5th iteration.

4.4 Case Study 1: Using Simple Network

4.4.1 Data and pricing strategy

Network

To test the modified TRANSIMS (v.3.0.1), a simple test network along with a managed-lane system was created. The test network is a 4-lane freeway in each direction and is shown in Figure 4.3. It includes two HOT lanes and two general purpose lanes (GPLs). In this dissertation, the "managed-lane system" refers to the tested HOT lane value pricing system. The “HOT lane” refers to the toll lane and the “GPLs” refer to the regular free lanes, respectively. HOVs can use the HOT lanes without paying a toll, while SOVs can use the HOT lanes by paying a toll. The test network also includes local streets (two outer lines in Figure 4.3). They are the alternative roads to the HOT lanes and GPLs. They are connected to and from the managed-lane system through access ramps. The test network is about 30 miles long and consists of 177 nodes and 210 links along with 14 activity locations. The managed-lane system is located in the middle of the freeway and is about 10 miles long. The HOT lanes are open to the east-bound in the morning-peak and to the west-bound in the evening-peak. It operates with 15-min dynamic varying toll rates, which is determined by V/C ratio on the HOT lanes.

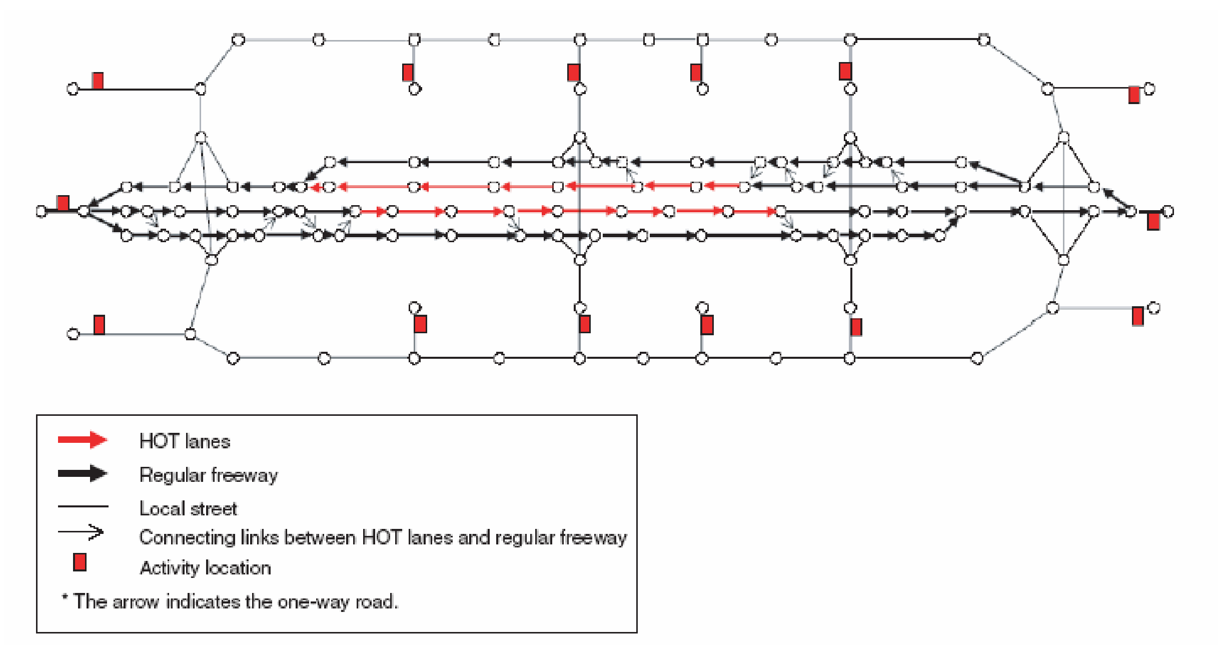

HOT lanes
Regular freeway
Local street
Connecting links between HOT lanes and regular freeway
Activity location
* The arrow indicates the one-way road.

Figure 4.3 Test network for case study 1.

The managed-lane system in this study is unique. It has only one entering point to the HOT lanes along with an intermediate exit point in the middle of the system, while GPLs have an additional entering point in the middle of the system. This system is very similar to the I-15 HOT lane value pricing system that is currently operating in California.

In I-15 HOT lane system, concrete barriers physically separate the HOT lanes from the GPLs. Contrary to the I-15 HOT lane system, the test network has one intermediate exit point to allow travelers to switch (exit) from the HOT lane to the GPLs through a connecting link. It is also important to note that the HOT lanes and the GPLs were separately configured in the test network to support link-by-link operation.

Demand

The demand was generated to simulate for morning peak period (from 6:30 to 8:30 a.m.). The demand includes a total of 13,026 travelers, and about 18% of them are HOVs (2,364 HOVs). The demand was generated to reflect the peak-demand (i.e., the highest traffic volume was generated during 7:15-7:30 a.m.).

Pricing Strategy

The dynamic tolls change every 15-min time interval, and are determined by V/C ratio in the first link of the starting HOT lanes. The higher the V/C ratio is in the lane, the higher the toll. In this case study, two different pricing decisions were tested. The pricing decision #1 was designed, so that the amount of update can increase by $0.50 (TOLL_INCREMENT), decrease by $0.25 (TOLL_DECREMENT), or maintain the rates as in the previous iteration. The pricing decision #2 implemented different values of TOLL_INCREMENT and TOLL_ DECREMENT. In this case, the incremental toll rates can be increased by $0.25, $0.50, $0.75 or $1.00 at maximum, depending on the difference of the old toll and the base toll. The values of TOLL_ DECREMENT were set by $0.25 or $0.50. A look-up table for dynamic toll decision was pre-defined, as shown in Table 4.7.

Table 4.7 Look-up Table for Toll Rate Decision

Range of V/C ratio	Toll Rate
V/C < 0.50	$ 0.00
0.51 < V/C < 0.55	$ 0.50
0.56 < V/C < 0.60	$ 1.00
0.61 < V/C < 0.65	$ 1.50
0.66 < V/C < 0.70	$ 2.00
0.71 < V/C < 0.75	$ 2.50
0.76 < V/C < 0.80	$ 3.00
0.81 < V/C < 0.85	$ 3.50
0.86 < V/C < 0.90	$ 4.00
V/C >= 0.90	$ 4.50

4.4.2 Experimental results

The eight simulation time slots were simulated to obtain 15-min dynamic tolls. As result of simulation, dynamic tolls were open for one hour and fifteen minutes (from 7:15 a.m. to 8:30 a.m.) in both pricing methods. When using the pricing decision #1, the tolls start from $0.25 (7:15-7:30 a.m.), gradually increase up to $1.75 (7:45-8:00 a.m.), and then gradually drop to $0.50 (8:15-8:30 a.m.). However, when using the pricing decision #2, the dynamic tolls fluctuate during toll hours (from $0.50, $1.25, $0.50, $1.00, to $0.25). The total revenue collected from SOVs traveling on the HOT lanes is $1,239.00 in the case of the pricing decision #1. In the case of the pricing decision #2, the total revenue is increased to $1,725.75 due to more SOVs choosing the HOT lanes. Apparently, the dynamic value pricing with the pricing decision #1 is preferred in terms of smooth change of tolls, however the pricing decision #2 is preferred if the interest is to make more profits from toll collections.

The output of each simulation time slot along with the 15-min dynamic toll rates (bold characters) in the case of pricing decision #1 is presented in Table 4.8. The table shows the change of toll rates throughout successive iterations within each simulation time slot. For instance, in simulation time slot #6, the Router tries to find out a toll rate for the time period of 7:45-8:00 a.m. The starting toll rate for the first iteration of this time slot is $1.50, which is obtained based on the previous output (simulation time slot #5). After routing all travelers in the first Router iteration, the Microsimulator gives a 15-min traffic volume on the HOT lane and V/C ratio of 0.32 is obtained. According to the look-up table, the base toll rate is $0.00 for this V/C ratio value. Thus, the new toll rate in the second iteration of the Router is reduced by $0.25 to $1.25. After the second iteration, the V/C ratio is 0.88, and the base toll rate is $4.00, resulting in a toll rate of $1.75 in the third iteration of the Router. Similarly, an updated toll rate after the fourth iteration is $1.50. However, the summation of total experienced travel times over all travelers after the fourth iteration does not decrease anymore from the previous iteration, so the program stops, and the final toll rate for the period 7:45-8:00 a.m. is decided at $1.75.

Table 4.8 Simulation Output

Simulation Time Slot 3 (7:00 - 7:15 a.m.)				
	Iteration 1	Iteration 2	Iteration 3	Iteration 4
Toll ($)	0.00	0.00	**0.00**	0.00
V/C ratio	0.35	0.36	0.37	0.36
Sum of total travel time (min)	9,172,819	9,703,666	9,221,691	9,903,981

Simulation Time Slot 4 (7:15 - 7:30 a.m.)				
	Iteration 1	Iteration 2	Iteration 3	
Toll ($)	0.50	**0.25**	0.75	
V/C ratio	0.33	0.63	0.66	
Sum of total travel time (min)	15,023,758	14,894,907	15,096,399	

Simulation Time Slot 5 (7:30 - 7:45 a.m.)				
	Iteration 1	Iteration 2	Iteration 3	Iteration 4
Toll ($)	0.75	0.50	**1.00**	1.00
V/C ratio	0.38	0.65	0.56	0.84
Sum of total travel time (min)	19,689,599	19,022,558	17,966,947	19,509,863

Simulation Time Slot 6 (7:45 - 8:00 a.m.)				
	Iteration 1	Iteration 2	Iteration 3	Iteration 4
Toll ($)	1.50	1.25	**1.75**	1.50
V/C ratio	0.32	0.88	0.52	0.28
Sum of total travel time (min)	23,103,068	22,395,246	22,147,832	24,483,994

Simulation Time Slot 7 (8:00 - 8:15 a.m.)				
	Iteration 1	Iteration 2	Iteration 3	
Toll ($)	1.50	**1.25**	1.00	
V/C ratio	0.23	0.24	0.24	
Sum of total travel time (min)	24,711,790	24,763,707	25,556,757	

Simulation Time Slot 8 (8:15 - 8:30 a.m.)				
	Iteration 1	Iteration 2	Iteration 3	Iteration 4
Toll ($)	1.00	0.75	**0.50**	0.50
V/C ratio	0.38	0.50	0.54	0.46
Sum of total travel time (min)	26,943,231	27,328,473	24,465,167	26,770,344

HOT Lane Choices by Travelers' Characteristics

Since TRANSIMS tracks all individual vehicle and person movements on second-by-second basis, we can identify each traveler's route choice. In the HOT lane value pricing system, HOVs can choose any route that has the least travel time since they do not have to pay tolls. On the other hand, SOVs are affected by tolls depending on their VOTs in choosing the HOT lanes vs. GPLs. We are particularly interested in analyzing the behavior of SOVs in response to dynamic tolls; specifically, the impact of tolls on the choice of the HOT lane based on characteristics of travelers (note that all analyses from this section are the results using the pricing decision #1). For this purpose, 4,158 travelers (828 HOVs and 3,330 SOVs) were obtained, who are eligible travelers for the choice between the HOT lanes and the GPLs during the toll hours (7:15-8:30 a.m.). During this time period, 42% of SOVs has chosen the HOT lanes, while 94% of HOVs has chosen the HOT lanes.

It is interesting to identify travelers categorized by VOTs in choosing between the HOT lanes and the GPLs. Numbers of SOVs grouped by VOT during the toll hours (7:15-8:30 a.m.) are depicted in Figure 4.4. The percentage of usage of HOT lanes by VOT categories is also shown in this figure (the percentage indicates percentage of travelers choosing the HOT lane within the same VOT category). The output is consistent with intuition: more travelers who have higher VOTs choose the HOT lanes (e.g., 69% out of 137 SOVs among those having VOT higher than \$80/h) than those have lower VOT (e.g., 18% out of 219 SOVs among those having VOTs less than \$10/h).

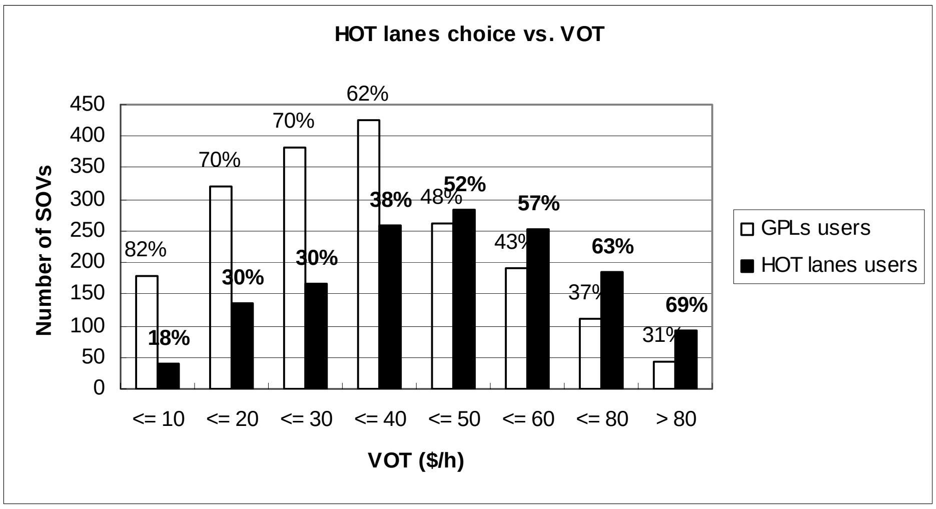


Figure 4.4 Comparison of HOT lane choice categorized by VOT.

The effects of different toll rates on SOVs with various VOTs are shown in Figures 4.5 through 4.9. The output during 8:00-8:15 a.m. (Figure 4.8) has a little different pattern, which can be explained as follows: 1) contrary to other time periods, all 141 HOVs chose the HOT lanes during 8:00-8:15 a.m., resulting in lower possibility for SOVs to choose the HOT lanes, and 2) the toll rate of $1.25 might be too high during this time which has too low V/C ratios as shown in Table 4.8. Nevertheless, all outputs show consistent results; that is, SOVs having higher VOTs are more likely to choose the HOT lanes.

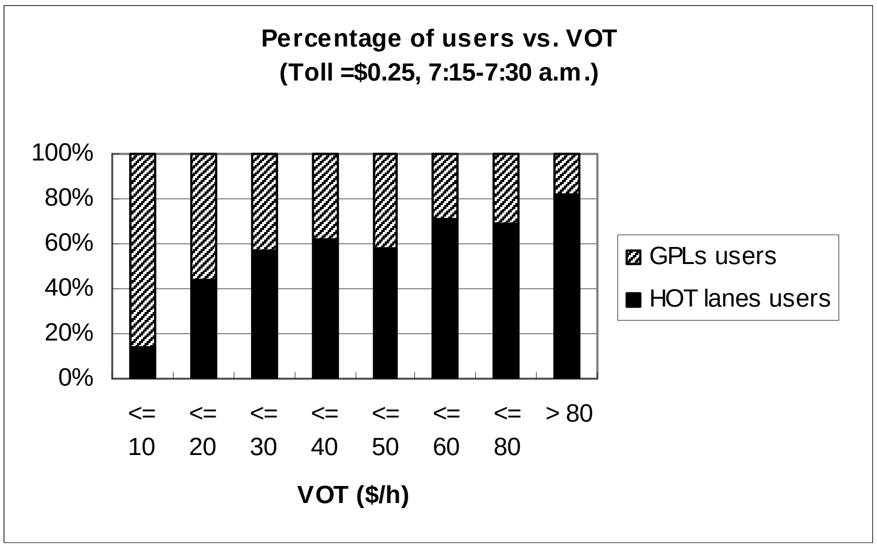


Figure 4.5 Sensitivity of HOT lane choice (toll = $0.25).

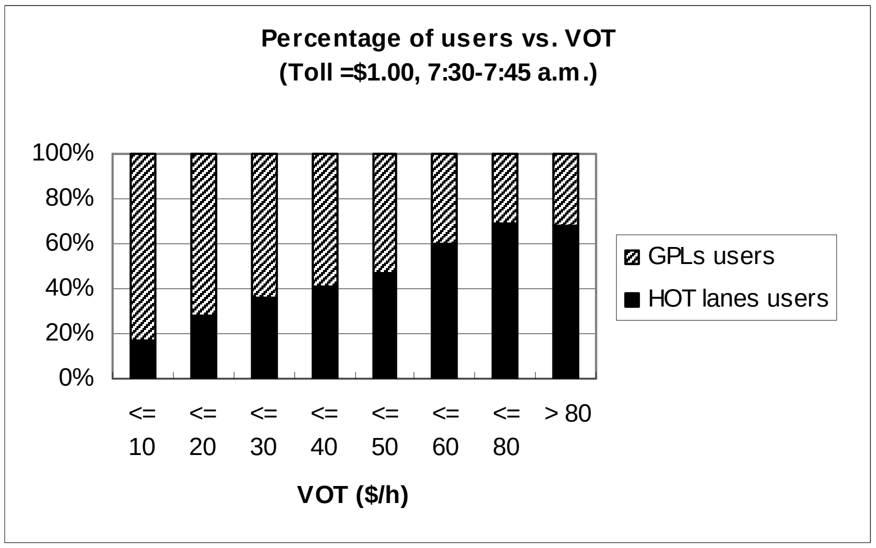


Figure 4.6 Sensitivity of HOT lane choice (toll = $1.00).

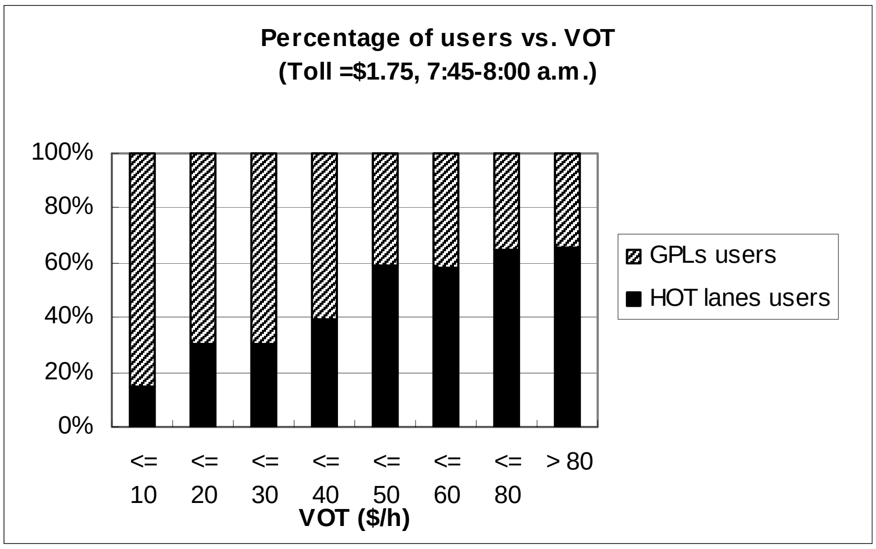


Figure 4.7 Sensitivity of HOT lane choice (toll = $1.75).

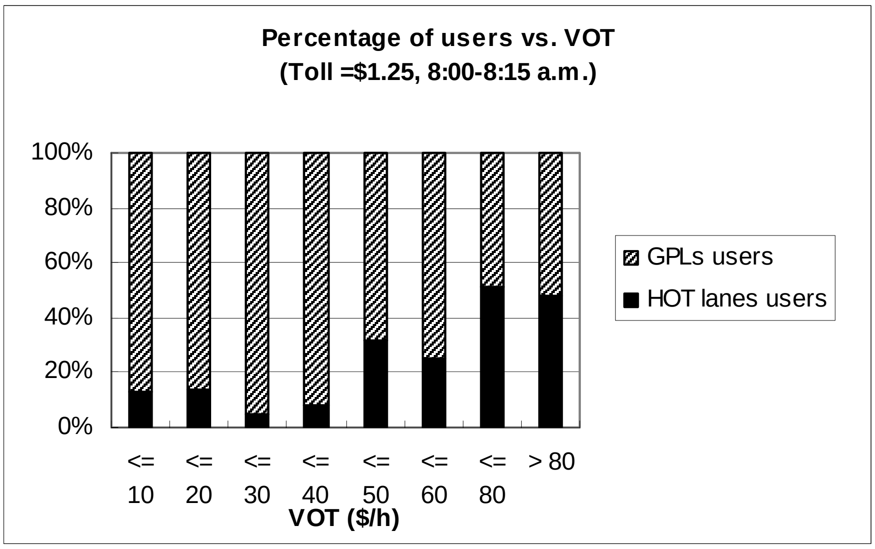


Figure 4.8 Sensitivity of HOT lane choice (toll = $1.25).

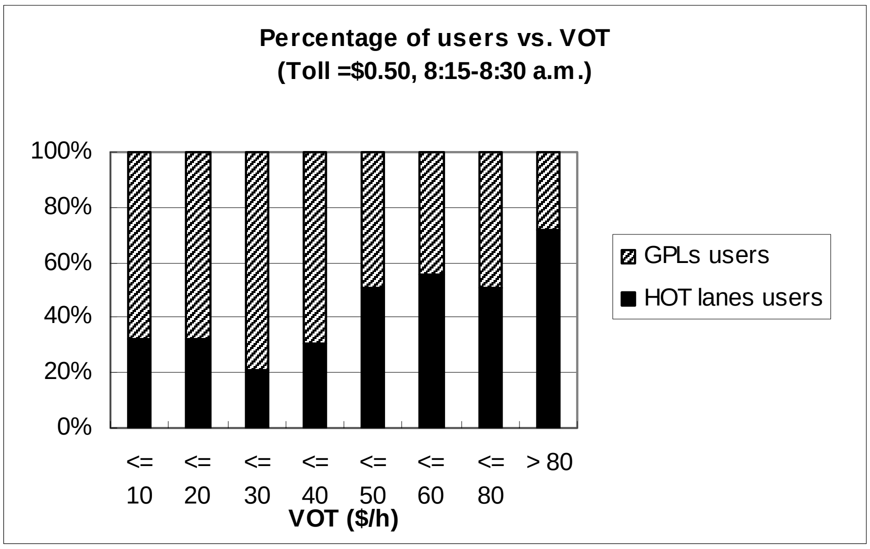


Figure 4.9 Sensitivity of HOT lane choice (toll = $0.50).

It is also worth analyzing the behavior of travelers based on different socio-economic characteristics in route choice between the HOT lanes and the GPLs. As TRANSIMS accepts various socio-economic characteristics of travelers, we can figure out how travelers respond to the different toll rates. Figures 4.10 through 4.13 (in the case of pricing decision #1) shows the HOT lane choice by SOVs based on income group, trip purpose, employment status, and age group, respectively. These figures illustrate that employees, who are at a higher income level, conducting a work trip, and are in the lower age group are most affected by toll rates when choosing between the HOT lanes and the GPLs.

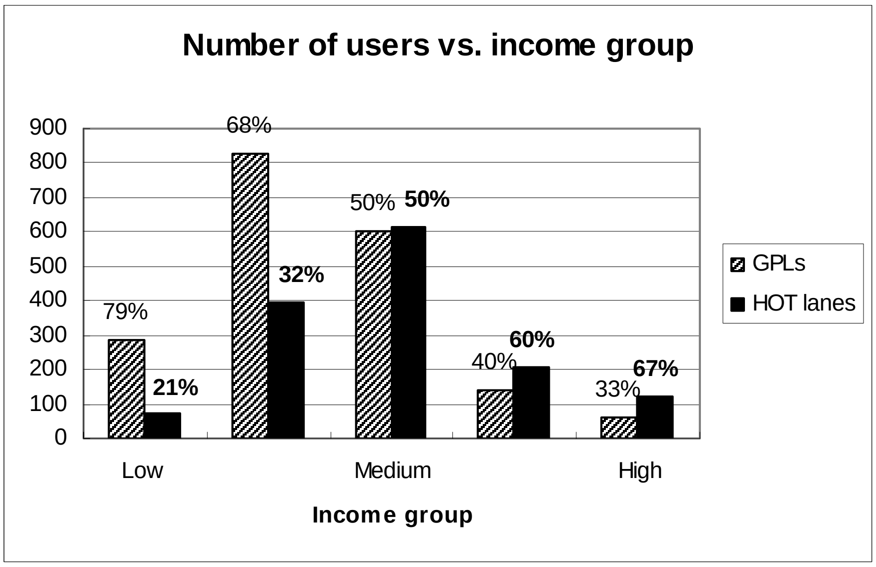


Figure 4.10 Comparison HOT lane choice by income group.

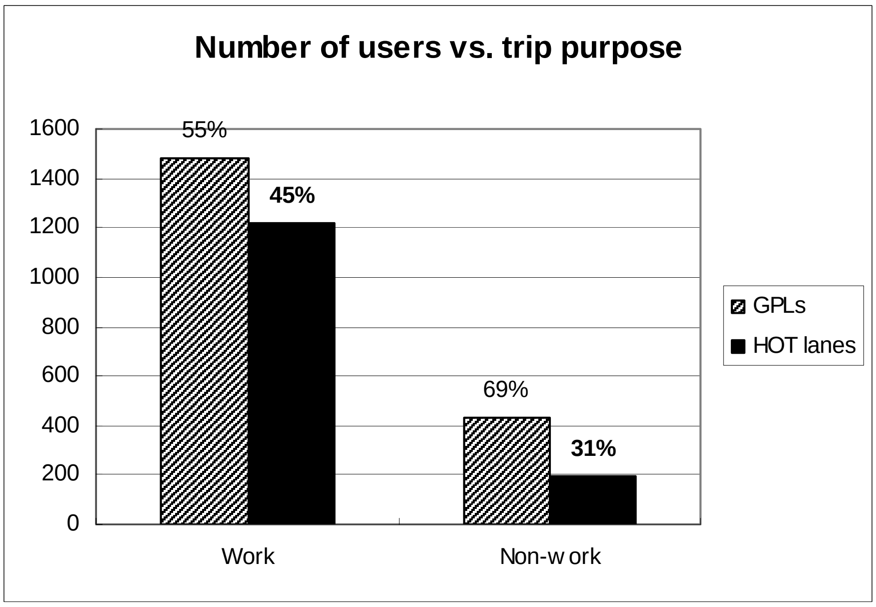


Figure 4.11 Comparison HOT lane choice by trip purpose.

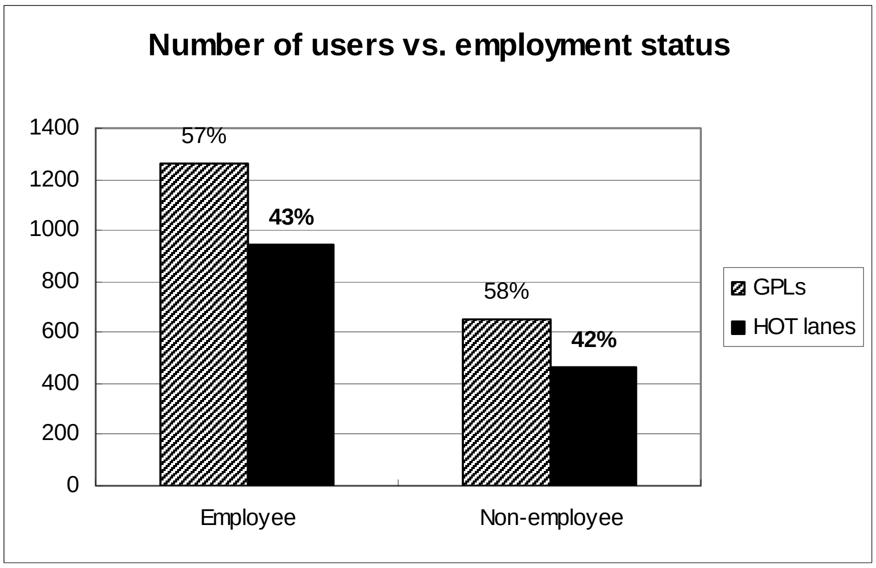


Figure 4.12 Comparison HOT lane choice by employment status.

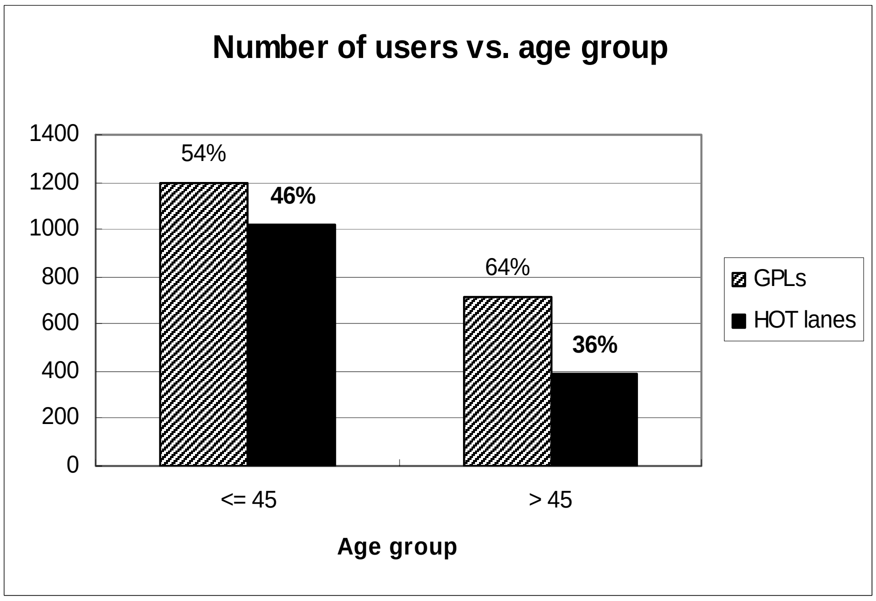


Figure 4.13 Comparison HOT lane choice by age group.

4.5 Case Study 2: Using Real-world Data from Portland, OR

4.5.1 Data and simulation scenarios

This case study is part of a FHWA research project. The objective of this research is to modify the source codes of new TRANSIMS version (v.4.0.1) to model and assess a HOT lane application using individual-based VOT function and 15-min dynamic toll rates. This version, which now works on both Linux and Windows, is different from the previous versions (e.g., v.3.0.1) in terms of file structures and how to run each module and/or program. Testing the HOT lane model on a real-world network is also part of the research objectives. The proposed HOT lane test site is a segment of I-5 northbound in Portland metropolitan region, Oregon. Some parallel arterial routes are available around I-5.

4.5.1.1 Portland network

Define study area

The available original Portland network contains a large area of the Portland, including total numbers of 8,375 links, 5,840 nodes and 26,794 activity locations. This large network requires a significant simulation run time. Accordingly, the original network was customized to get a sub-area network. The demand and the supply applicable to that sub-area were accordingly extracted. Defining the size of the transportation network is important at this point, because it affects data extraction from input data (e.g., activity, household, person and vehicle files). There are some advantages and disadvantages in terms of size of network. If the network is relatively small, we could save simulation time to run TRANSIMS. On the other hand, if it is too small, we have to make up lots of itinerant trips that potentially use the HOT lane. The study area along with the original Portland network, including major freeways, is shown in Figure 4.14. The study area is depicted in yellow and the proposed location of the HOT lane is shown in red line. The study area was determined by the fact that the proposed segment of I-5 HOT lane application is located from Hwy 217 to I-405 near the Central Business District (CBD). The study area provides enough coverage to include most of the potential users of the HOT lane.

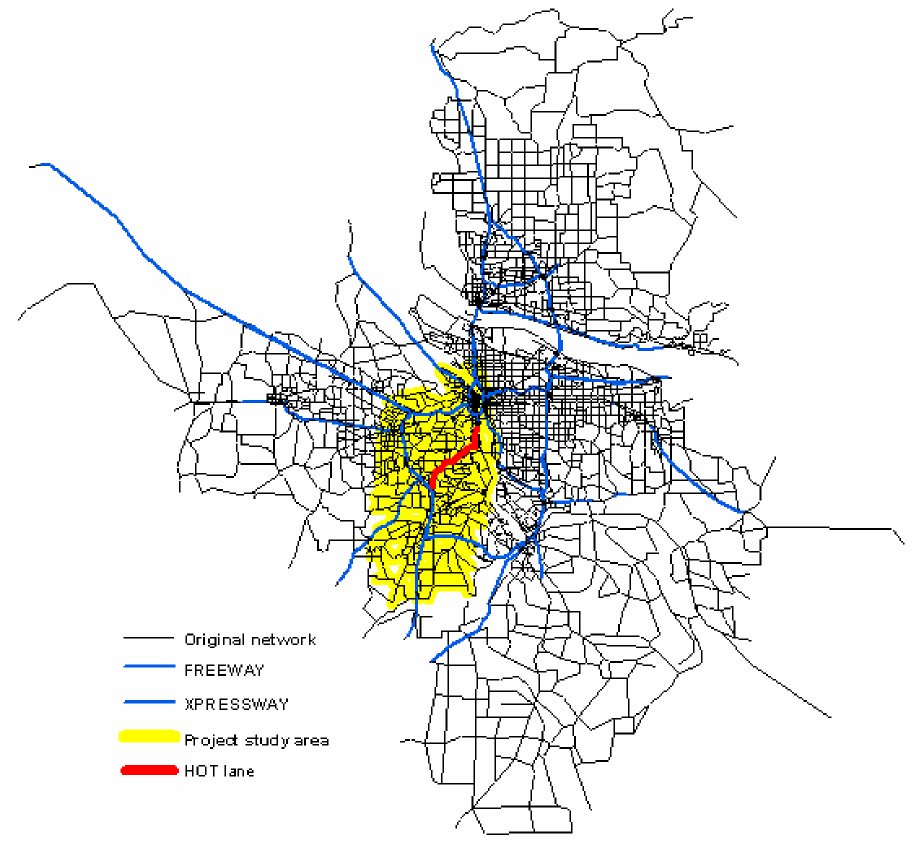


Figure 4.14 Portland network and study area.

Once the study area was defined, those links that fall outside of the sub area were cut off. After conducting lots of tests to fix the network problem (e.g., lane connectivity, signal problem, etc.) during the extraction of original network, the final study area of the Portland network was obtained. Table 4.9 shows the comparison of the size of network. The test network covers a large urbanized area, including 1, 657nodes, 2, 336links and 5,510 activity locations. This is about 30% of the original network in terms of numbers of links, nodes and activity locations.

Table 4.9 Comparison of Original Portland Network with Sub-are Network

	Original Network	Sub-are Study Network
Number of links	8,375	2,336
Number of nodes	5,840	1,657
Number of activity locations	26,794	5,510

Modifying network to add HOT lane

The existing I-5 northbound freeway has 3 GPLs without a HOV lane or HOT lane. One lane of the existing 3 GPLs was converted to the HOT lane using ArcGIS, so that the managed-lane system includes the 1 lane of the HOT lane and the 2 lanes of the GPLs. The system is 6.2 miles long, and is located from Hwy 217 (Exit 292) to I-405 (Exit 298) near the CBD. Figure 4.15 shows the study area along with the location of the managed-lane system and the primary freeways. The HOT lane is available in northbound. The managed-lane system in this test network is unique. It has only one entering point to the HOT lane along with three intermediate exit points in the middle of the system, while GPLs have multiple entering/exit points. The system is very similar to the I-15 HOT lane value pricing system that is currently operating in California. Contrary to the I-15 HOT lane system, the test network in this research has three intermediate exit points to allow travelers to switch (exit) from the HOT lane to the GPLs through a connecting link. It is also important to note that one HOT lane and two GPLs were separately configured in the test network.

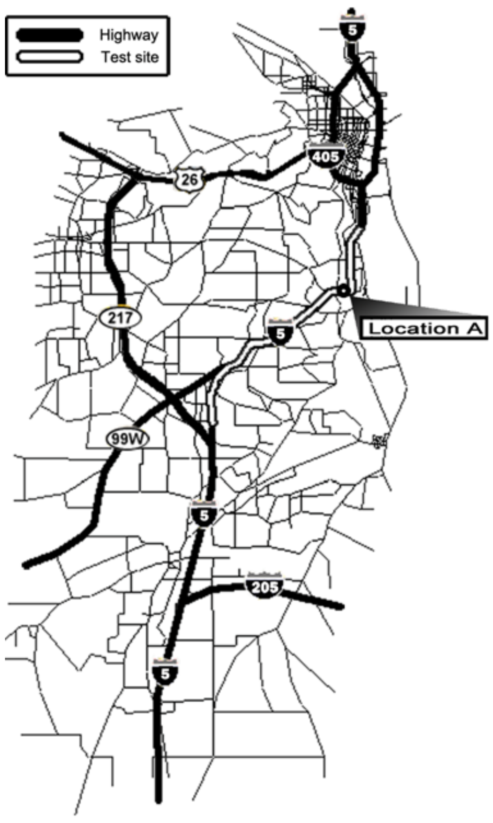


Figure 4.15 Test network with the location of the managed-lane system.

4.5.1.2 Portland activity data

Based on the available input data (i.e., survey data, network files), the ActGen program was run to obtain the Portland activity file. This activity file and the corresponding files (i.e., household population, and vehicles files) were suppressed and cleaned up for this research.

Step 1: Collect activities relevant to this study

First, the following activities were excluded because they are outside the scope this study.

- Excluding activities made after 10 a.m.

This study focuses on the analysis for the morning peak period. Thus, the activity data made after 10 a.m. were suppressed.

- Excluding some mode types

The activity data excludes those types of mode, including bus, rail, and park-&-ride because the complete input data for the transit was not available yet.

Step 2: Identify activities by internal-internal, internal-external, and external-internal

The home location of each person can be identified from the household file. In addition, we can identify activity locations inside the study area from the Activity_Location file of network. The activities by internal-internal and internal-external were identified and categorized.

- Internal-internal activities include people whose all activities are made within the study area.
- Internal-external activities include people who live within the study area, and at least one of his/her activities is made outside of the study area.

The data of external-internal activities and external-external activities were removed from the activity file in this research. Accordingly, the data includes a person who makes a trip starting from internal zones, while a person's activities, whose household is located out of the study area, were ignored.

Step 2.1: Change location ID for external data

Once external activity locations IDs of internal-external activities are identified in Step 2 above, they were changed to the activity location ID in one of the closest internal locations.

Step 2.2: Suppressing data

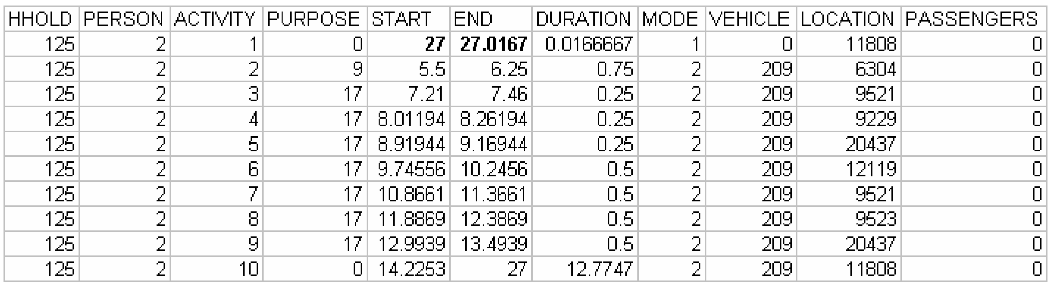
If the two consecutive activities are in both external locations, the remaining data from the second external activities were removed. This was necessary, because if the two consecutive activities are both in external locations, then the traveler's remaining trips can be distorted in the output.

Step 3: Suppressing/modifying unreasonable data

During the test of extracted input data, some unreasonable/unnecessary data in the original activity file were identified, and they were fixed.

- Adjusting start time

Some travelers have the start time of their first activity after midnight (i.e., 24.xx, 25.xx, etc.) as shown in Figure 4.16. This is caused by the fact that activity file is obtained from census data and activity survey data, which is based on 27 hours of the day. To avoid any further problems, the START time for those travelers of having the starting/ending time after 24 was adjusted. For example, if a traveler's starting time of his/her first activity is 25.30, the start time was changed to 1.30.

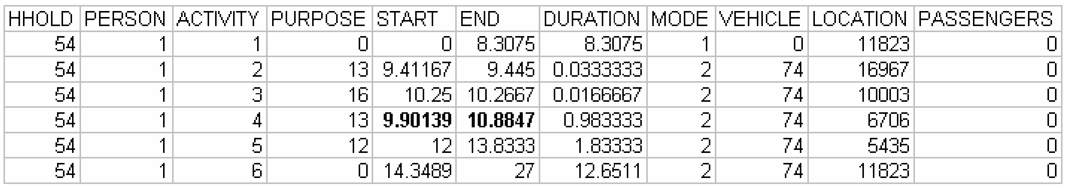


HHOLD	PERSON	ACTIVITY	PURPOSE	START	END	DURATION	MODE	VEHICLE	LOCATION	PASSENGERS
125	2	1	0	**27**	**27.0167**	0.0166667	1	0	11808	0
125	2	2	9	5.5	6.25	0.75	2	209	6304	0
125	2	3	17	7.21	7.46	0.25	2	209	9521	0
125	2	4	17	8.01194	8.26194	0.25	2	209	9229	0
125	2	5	17	8.91944	9.16944	0.25	2	209	20437	0
125	2	6	17	9.74556	10.2456	0.5	2	209	12119	0
125	2	7	17	10.8661	11.3661	0.5	2	209	9521	0
125	2	8	17	11.8869	12.3869	0.5	2	209	9523	0
125	2	9	17	12.9939	13.4939	0.5	2	209	20437	0
125	2	10	0	14.2253	27	12.7747	2	209	11808	0

Figure 4.16 Example of activity start time.

- Problem of unreasonable order of activities

There were a few unreasonable activity patterns (i.e., reversed start time of the two activities) in the original file. For example, in Figure 4.17 the start time of a traveler's activity number 3 is 10.25 and the subsequent start time of the activity number 4 is 9.90. This does not make sense. Thus, the subsequent activities (i.e., from activity number 4) of this traveler were removed from the activity file.

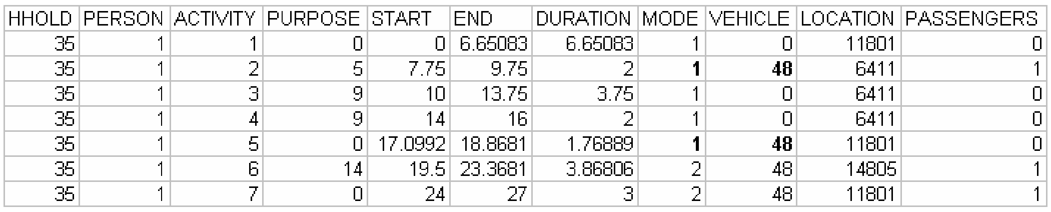


HHOLD	PERSON	ACTIVITY	PURPOSE	START	END	DURATION	MODE	VEHICLE	LOCATION	PASSENGERS
54	1	1	0	0	8.3075	8.3075	1	0	11823	0
54	1	2	13	9.41167	9.445	0.0333333	2	74	16967	0
54	1	3	16	10.25	10.2667	0.0166667	2	74	10003	0
54	1	4	13	**9.90139**	**10.8847**	0.983333	2	74	6706	0
54	1	5	12	12	13.8333	1.83333	2	74	5435	0
54	1	6	0	14.3489	27	12.6511	2	74	11823	0

Figure 4.17 Example of unreasonable order of activity.

- Problem of unreasonable mode type

There were a few unreasonable mode types. A walk mode (mode number 1 in the activity file) should not have a vehicle number in the VHHICLE field in the activity file. However, some activities had a non-zero vehicle number even though the activity is made by walk (Figure 4.18). Thus, those data was fixed, so that mode type was change to 2 (auto) if a vehicle exists.



HHOLD	PERSON	ACTIVITY	PURPOSE	START	END	DURATION	MODE	VEHICLE	LOCATION	PASSENGERS
35	1	1	0	0	6.65083	6.65083	1	0	11801	0
35	1	2	5	7.75	9.75	2	**1**	**48**	6411	1
35	1	3	9	10	13.75	3.75	1	0	6411	0
35	1	4	9	14	16	2	1	0	6411	0
35	1	5	0	17.0992	18.8681	1.76889	**1**	**48**	11801	0
35	1	6	14	19.5	23.3681	3.86806	2	48	14805	1
35	1	7	0	24	27	3	2	48	11801	1

Figure 4.18 Example of unreasonable mode type.

Step 4: Extracting Household, Population and Vehicle data

Based on information (HH id and Person id) from the extracted activity file, the corresponding household, population and vehicle files were extracted.

- Household, population and vehicle files were extracted by matching HH id and person id based on the extracted activity file.
- The external location ID in the household and vehicle files was also changed accordingly.

Step 5: Finalizing

As results of previous steps, the final testing data included a total of 120,080 households, 240,982 travelers (average 2 travelers/household), and 503,801 activities (average 2.09 activities/traveler).

4.5.1.3 Simulation scenarios

The two scenarios were run and tested using the data from Portland, OR. The base scenario was run on the existing network. In this network, 3 GPLs are available in I-5 freeway without a HOV/HOT facility and any tolls. The alternative scenario was run on the modified network by including the managed-lane system, consisting of 1 HOT lane and 2 GPLs. The individual VOT and the 15-min dynamic tolls were implemented in this alternative scenario.

4.5.2 Experimental results

Overall network performance

The network performances in the overall network were slightly improved in the alternative scenario, as shown in Table 4.10. Although vehicle-miles traveled (VMT) was not much improved in the alternative case, vehicle-hours traveled (VHT) and hours of delay (HOD) were improved by 3.32 % and 7.81 %, respectively.

Table 4.10 Comparison of Overall Network Performance

	Base case	Alternative case (dynamic toll)	Percentage of reduction from Base case
Veh-miles traveled	1,129,889	1,128,021	0.17%
Veh-hours traveled	48,666	47,052	3.32%
Hours of delay	19,247	17,744	7.81%

15-min Dynamic Tolls

The alternative scenario involved 10 simulation time slots to find 15-min dynamic tolls that vary by V/C ratio on the HOT lane. Multiple simulation runs were also involved in each time slot. The two scenarios were simulated using a Linux machine with 500-MHZ CPU without parallel processing. The CPU time in the base case scenario was 6 hours and 23 minutes. The CPU time in the alternative case scenario consumed more time with 7 hours and 20 minutes. If a parallel processing is used, the CPU time would be improved. Table 4.11 summarizes the number of Microsimulator iterations in each time slot and the final toll rates for each 15-min time period. As result of the simulation, tolls were effective during the time periods of 6:00-6:15, 6:30-6:45, 7:00-7:15, 7:15-7:30 and 7:30-7:45 a.m. During the toll periods, about 10% of travelers were HOVs among the HOT lane users. The total revenue collected from SOVs was $2,864/day (in the morning peak). It is equivalent to $744,640 per year based on 5 days a week and 52 weeks per year. The maximum toll was $4 during 7:00-7:15 a.m. After 7:15 a.m., the tolls were decreased gradually until 7:45 a.m. It was interesting that once the toll starts at 6:00 a.m., the tolls did not gradually increase until 7:00 a.m. This is because the demand on the managed-lane system was not steadily increased, and it was rapidly varied during 6:00-7:00 a.m. Even though this is a little surprise, it precisely illustrates the success and need of the dynamic tolls that vary by level of service (i.e., V/C ratio). If demands are very steady by time-of-day and are quite predictable, then policy-makers might not need to consider the dynamic tolls. Rather, they might be interested in implementing pre-determined (fixed) time-dependent tolls. The dynamic toll application will be more effective tool in the place where demand is not steady by time-of-day or where travelers are sensitive to tolls in choosing their routes. In a real situation, this unexpected toll change between 6:45-7:00 a.m. and 7:00-7:15 a.m. will smooth out to avoid abrupt toll change.

Table 4.11 Final 15-min Dynamic Tolls

Time slot	Msim iteration #	Toll period (a.m.)	Final toll ($)
1	9	N/A	N/A
2	5	6:00-6:15	1.50
3	8	6:15-6:30	0.00
4	6	6:30-6:45	0.50
5	8	6:45-7:00	0.00
6	10	7:00-7:15	4.00
7	7	7:15-7:30	2.25
8	5	7:30-7:45	1.00
9	8	7:45-8:00	0.00
10	11	N/A	N/A

Tables 4.12 through 4.17 present how 15-min dynamic tolls are decided iteration-by-iteration for six time slots (from 6:15 to 7:45 a.m.). Note that dynamic toll update for

the time period from 6:00 a.m. to 6:15 a.m. was shown in Table 4.6. In each time slot, the old toll in the first iteration of the Router is obtained the dynamic toll from the previous time slot. For example, as shown in Table 4.13, 15-min toll rate for the time period from 6:30-6:45 a.m. (simulation time slot #4) was decided at \$0.50. This toll rate is now the old toll for the first iteration of the Router in time slot #5 that is shown in Table 4.14. It should be noted that the tolls rates were effective and gradually decreased during the time periods from 7:15 to 7:45 (time slots # 7 and 8), even though the traffic volumes are not very high. This is because the toll rate change was designed to prevent abrupt toll rate change between the two successive 15-min dynamic tolls and to change tolls smoothly.

Table 4.12 Dynamic Toll Updates for Time Slot #3 (6:15-6:30 a.m.)

Iteration number	V/C ratio	Base toll (\$)	Old toll (\$)	Updated toll (\$)	Summation of *ET* (hrs) from Msim
1	0.358	0.00	1.50	1.25	39,649
2	0.330	0.00	1.25	1.00	38,711
3	0.366	0.00	1.00	0.75	38,461
4	0.394	0.00	0.75	0.50	37,419
5	0.354	0.00	0.50	0.25	37,624
6	0.381	0.00	0.25	0.00	37,325
7	0.373	0.00	0.00	0.00	36,910
8	0.389	0.00	0.00	**0.00**	36,769
9	0.364	0.00	0.00	0.00	36,974

Table 4.13 Dynamic Toll Updates for Time Slot #4 (6:30-6:45 a.m.)

Iteration number	V/C ratio	Base toll (\$)	Old toll (\$)	Updated toll (\$)	Summation of *ET* (hrs) from Msim
1	0.650	1.00	0.00	0.50	39,083
2	0.587	0.50	0.50	0.50	38,251
3	0.636	1.00	0.50	1.00	38,330
4	0.587	0.50	1.00	0.75	37,551
5	0.632	1.00	0.75	0.75	37,689
6	0.581	0.50	0.75	**0.50**	36,730
7	0.636	1.00	0.50	1.00	37,305

Table 4.14 Dynamic Toll Updates for Time Slot #5 (6:45-7:00 a.m.)

Iteration number	V/C ratio	Base toll ($)	Old toll ($)	Updated toll ($)	Summation of *ET* (hrs) from Msim
1	0.577	0.50	0.50	0.50	38,721
2	0.495	0.00	0.50	0.25	39,276
3	0.541	0.00	0.25	0.00	40,305
4	0.509	0.00	0.00	0.00	38,304
5	0.562	0.50	0.00	0.50	38,540
6	0.491	0.00	0.50	0.25	37,707
7	0.570	0.50	0.25	0.25	37,229
8	0.537	0.00	0.25	**0.00**	36,024
9	0.566	0.50	0.00	0.50	36,514

Table 4.15 Dynamic Toll Updates for Time Slot #6 (7:00-7:15 a.m.)

Iteration number	V/C ratio	Base toll ($)	Old toll ($)	Updated toll ($)	Summation of *ET* (hrs) from Msim
1	0.890	3.50	0.00	0.50	39,213
2	0.697	1.50	0.50	1.00	40,573
3	0.853	3.50	1.00	1.50	42,338
4	0.806	3.00	1.50	2.00	43,909
5	0.878	3.50	2.00	2.50	41,099
6	0.952	4.00	2.50	3.00	38,522
7	0.966	4.00	3.00	3.50	38,326
8	0.930	4.00	3.50	4.00	37,562
9	0.952	4.00	4.00	4.00	37,446
10	0.966	4.00	4.00	**4.00**	36,406
11	1.000	4.00	4.00	4.00	37,003

Table 4.16 Dynamic Toll Updates for Time Slot #7 (7:15-7:30 a.m.)

Iteration number	V/C ratio	Base toll ($)	Old toll ($)	Updated toll ($)	Summation of *ET* (hrs) from Msim
1	0.375	0.00	4.00	3.75	38,749
2	0.370	0.00	3.75	3.50	37,934
3	0.448	0.00	3.50	3.25	37,683
4	0.581	0.50	3.25	3.00	37,177
5	0.491	0.00	3.00	2.75	37,643
6	0.490	0.00	2.75	2.50	37,221
7	0.560	0.50	2.50	**2.25**	36,533
8	0.514	0.00	2.25	2.00	36,884

Table 4.17 Dynamic Toll Updates for Time Slot #8 (7:30-7:45 a.m.)

Iteration number	V/C ratio	Base toll ($)	Old toll ($)	Updated toll ($)	Summation of *ET* (hrs) from Msim
1	0.463	0.00	2.25	2.00	39,362
2	0.299	0.00	2.00	1.75	39,415
3	0.349	0.00	1.75	1.50	42,229
4	0.305	0.00	1.50	1.25	42,141
5	0.253	0.00	1.25	**1.00**	39,606
6	0.010	0.00	1.00	0.75	40,217

Volume comparison

In this unique managed-lane system, traffic volumes in the HOT lane are decreasing on the links approaching downtown because there is only one entering point, and some travelers are leaving the HOT lane through the provided exits. On the other hand, traffic volumes in the GPLs are increasing on the links toward downtown because additional travelers are entering the GPLs in the middle of the system. Figure 4.19 shows the comparison of traffic volumes (veh/hr/lane) at location A (see Figure 4.15 for this location) between the HOT lane and GPLs. As expected, higher volumes were observed in the GPLs than in the HOT lane.

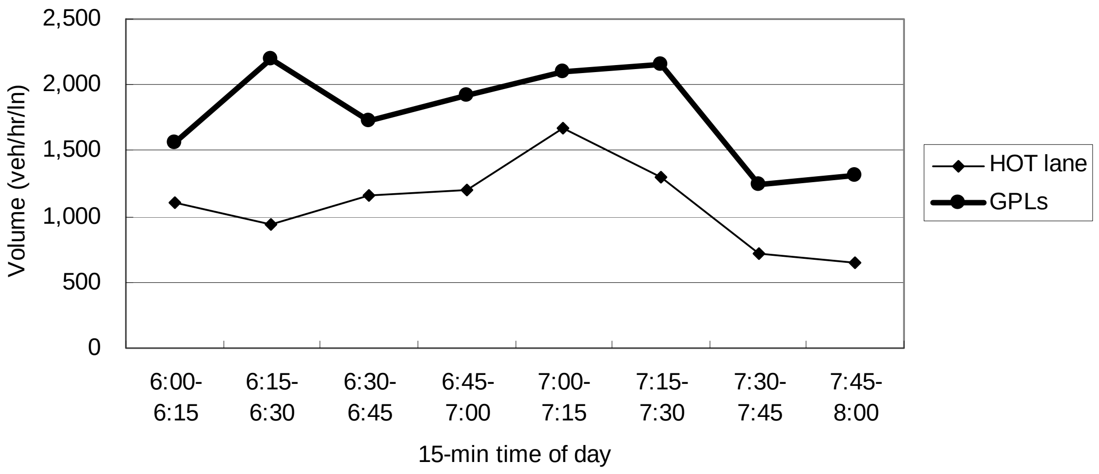
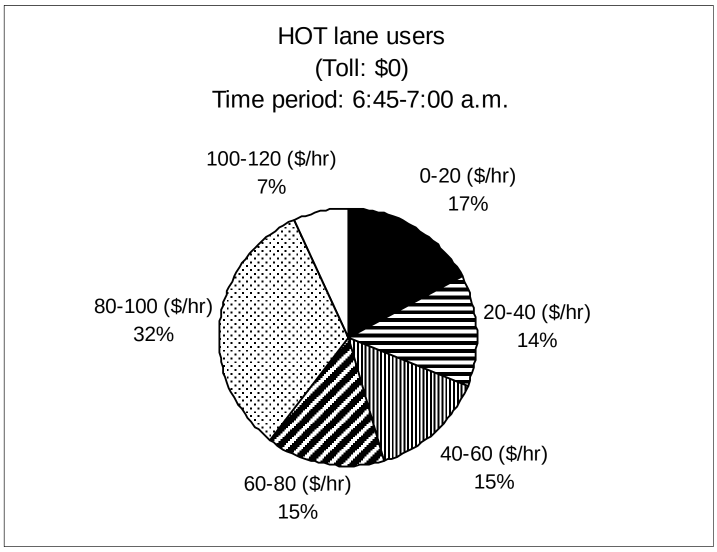


Figure 4.19 Volume comparison at location A between the HOT lane and GPLs.

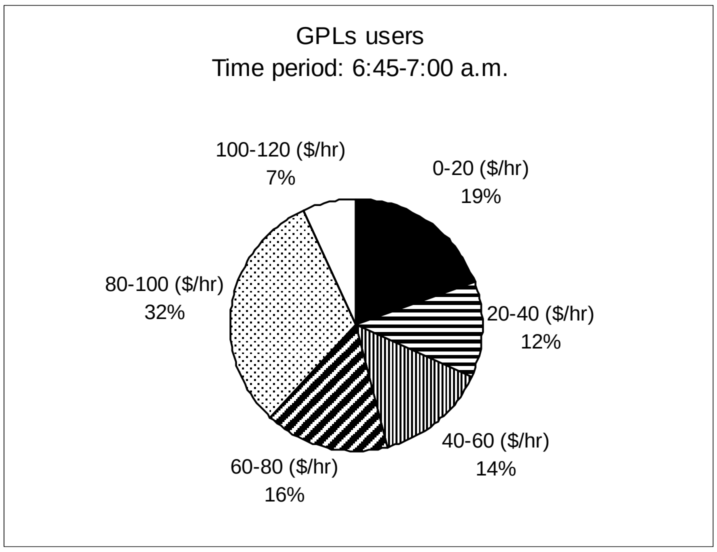
Impacts of tolls by VOT groups on route choice (HOT vs. GPLs)

Figures 4.20 through 4.22 present the VOT comparisons categorized by 6 VOT groups between the HOT lane users and the GPLs users. When there is no toll, VOTs of the travelers in both the HOT lane and the GPLs show fairly similar patterns in all 6 VOT categories (Figure 4.20(a) vs. Figure 4.20(b)). It makes sense because VOTs do not impact on route choices in the generalized travel time function when there is no toll. It is interesting, however, that the highest proportion (32 %) out of the 6 VOT groups is travelers who have VOTs of between 80$/hr and 100 $/hr. These results can be attributed to travelers commuting from the suburbs to downtown during peak periods, who represent high-income households (i.e., high-income households mean that they have high VOTs). When tolls are effective, the outputs were also consistent with intuition: that is, more travelers with higher VOTs choose the HOT lane than those with lower VOT. For example, as shown in Figure 4.21, when toll is $4.00 (during 7:00-7:15 a.m.), about 65 % of the HOT lane users belong to higher VOT groups (60-120 $/hr). On the other hand, the majority VOT group in traveling on the GPLs was the lowest VOT group (30 % of travelers belong to VOT with 0-20 $/hr). During 7:15-7:30 a.m. (toll is $2.25), about 70 %

of the HOT lane user belong to higher VOT groups (60-120 $/hr), while about 70 % of the GPLs users belong to lower VOT groups (0-60 $/hr) (Figure 4.22).

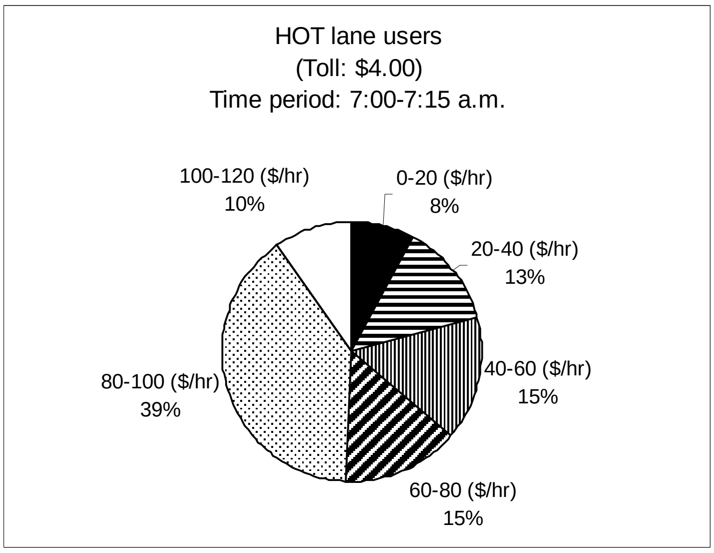


(a)

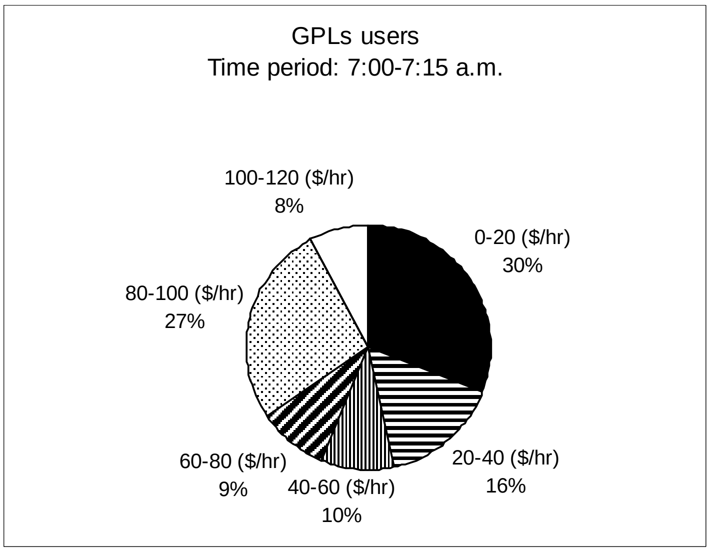


(b)

Figure 4.20 VOT comparison between HOT lane users (a) and GPLs users (b) during 6:45-7:00 a.m. (no toll).

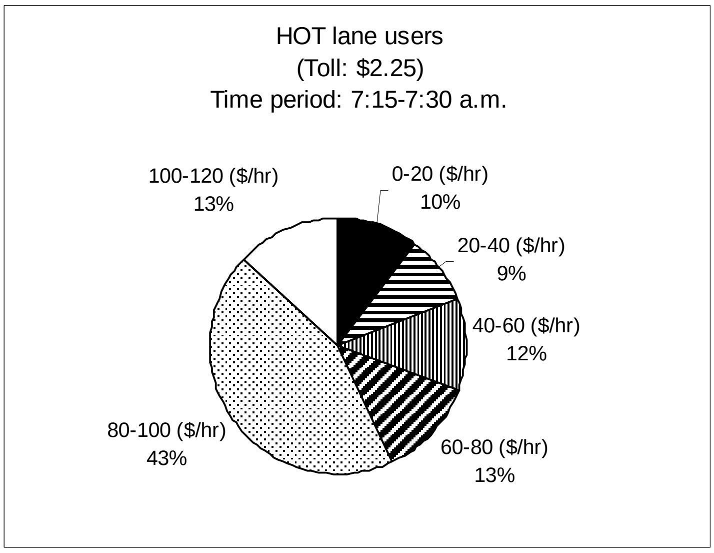


(a)

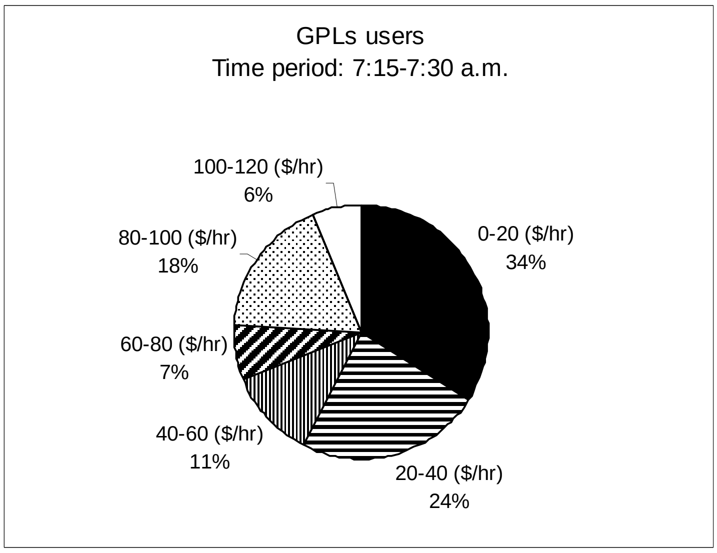


(b)

Figure 4.21 VOT comparison between HOT lane users (a) and GPLs users (b) during 7:00-7:15 a.m. (toll: $4.00).



(a)

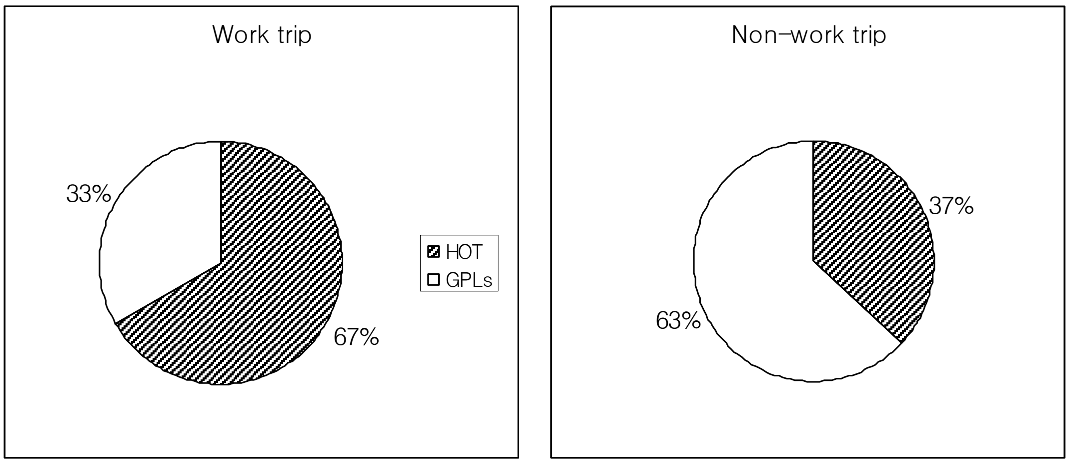


(b)

Figure 4.22 VOT comparison between HOT lane users (a) and GPLs users (b) during 7:15-7:30 a.m. (toll: $2.25).

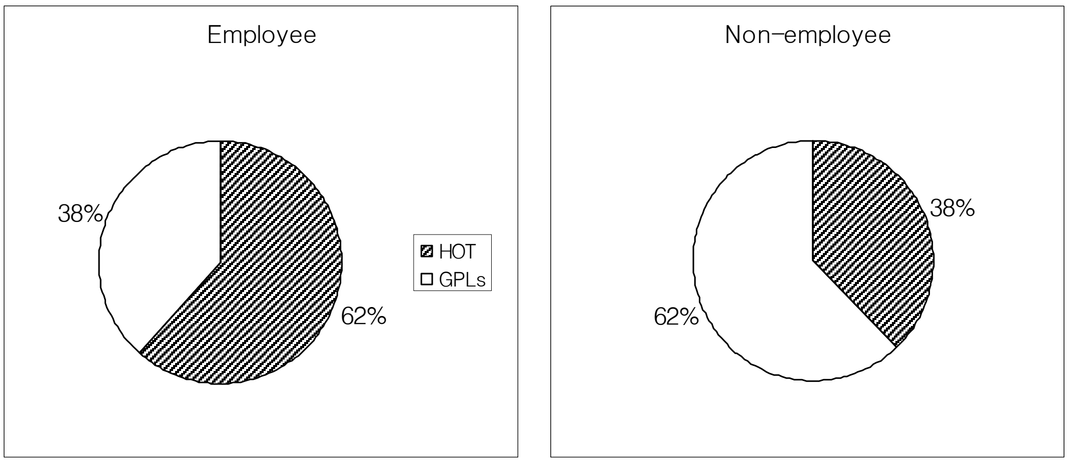
Route choice (HOT vs. GPLs) by travelers' characteristics

TRANSIMS maintains the identity of the traveler throughout the simulation and is capable of accessing the database of each individual. It allows identifying how travelers respond to the tolls. It is interesting to analyze impacts of individual/trip characteristics on route choice between the HOT lane and the GPLs. Figures 4.23 through 4.26 show the comparison of route choice between the HOT lane and the GPLs by trip purpose (Figure 4.23), employment status (Figure 4.24), age group (Figure 4.25) and income group (Figure 4.26), respectively. These figures represent the travelers, who enter into the start link of the HOT lane and the corresponding GPLs, when toll is the highest ($4.00) from 7:00 to 7:15 a.m. The figures illustrate that trip purpose, employment status and income level obviously affect the choice between the HOT lane and the GPLs. For example, although most of travelers on both the HOT lane and the GPLs have work trip purpose, the majority of work trip travelers (67 %) choose the HOT lane, while the majority of non-work travelers (63%) choose the GPLs.



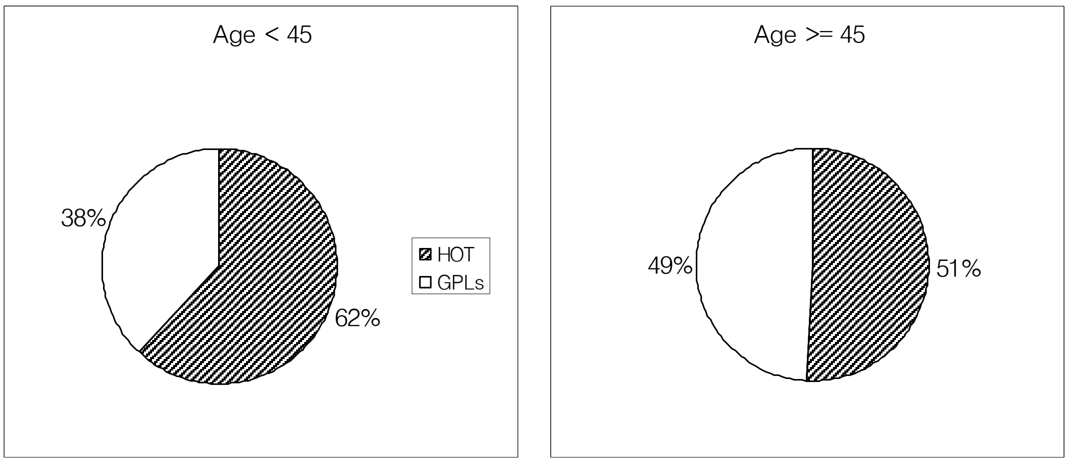
(a) (b)

Figure 4.23 Route choice by trip purpose.



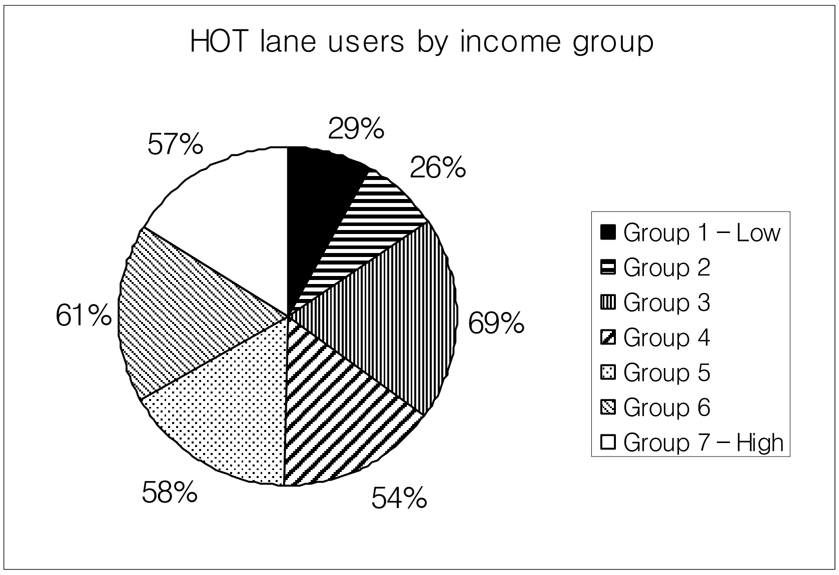
(a) (b)

Figure 4.24 Route choice by employment status.

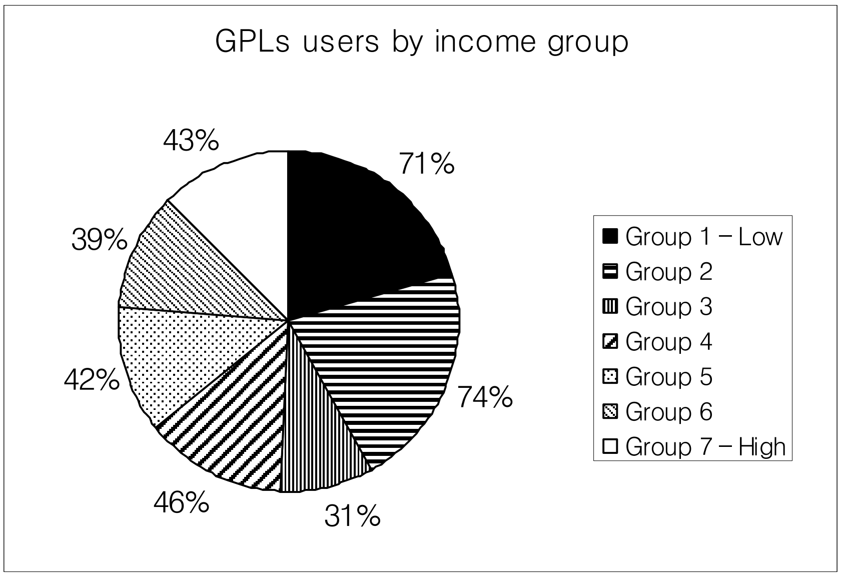


(a) (b)

Figure 4.25 Route choice by age group.



(a)



(b)

Figure 4.26 Comparison of HOT lane users (a) and GPLs users (b) by income group.

Comparison of trip travel times

Other interesting result includes the comparison of trip travel times between the base and the alternative scenarios. This is particularly interesting because we can identify why some travelers choose the HOT lane by paying tolls and how much they save their travel times. For this purpose, twenty travelers were randomly selected, who travel on the HOT lane during the highest toll rate ($4.00) and have high travel time savings. Table 4.18 presents the travel time difference of the selected travelers between the two scenarios. The table also presents travelers' socio-economic and trip characteristics, and their VOTs. Among the selected travelers, household ID 102291 and person ID 1 had the highest travel time saving (about 32 minutes).

Table 4.18 Travel Time Comparison between Base and Alternative Scenarios

Household ID	Person ID	Trip ID	Travel Time in Base Case (min)	Travel Time in Alternative Case (min)	**Travel Time Difference (Base-Alt) (min)**	VOT ($/hr)	Age	Income Group (1-low; 7-high)	Employ-ment Status (1-yes; 0-no)	Trip Purpose (1-work; 0-others)
18567	1	2	19.33	14.68	**4.65**	82.50	59	6	1	1
22806	1	2	18.73	13.22	**5.52**	91.25	34	6	1	1
89569	2	2	37.22	21.05	**16.17**	42.90	25	3	1	1
94804	1	4	38.78	19.83	**18.95**	6.67	77	4	0	0
99967	1	2	42.40	23.50	**18.90**	17.23	37	2	0	0
101496	2	2	21.70	15.70	**6.00**	51.25	36	4	1	1
101940	1	2	41.13	17.10	**24.03**	71.25	26	5	1	1
102291	1	2	59.07	27.10	**31.97**	71.25	38	5	1	1
107968	1	2	30.18	20.48	**9.70**	62.50	50	5	1	1
108188	2	2	43.23	20.20	**23.03**	91.25	33	6	1	1
295031	2	2	19.80	15.17	**4.63**	34.58	23	2	1	1
411867	1	2	31.92	14.85	**17.07**	71.25	37	5	1	1
412512	2	2	28.15	12.27	**15.88**	91.25	31	6	1	1
413886	1	2	32.53	15.22	**17.32**	111.25	32	7	1	1
415355	1	4	21.82	14.62	**7.20**	20.57	29	5	1	0
415435	2	2	24.50	14.23	**10.27**	71.25	31	5	1	1
416577	2	2	24.27	15.95	**8.32**	51.25	20	4	1	1
417735	1	2	28.17	17.40	**10.77**	91.25	37	6	1	1
481625	2	2	35.20	21.33	**13.87**	91.25	26	6	1	1
484837	2	2	43.58	24.50	**19.08**	62.50	52	5	1	1

Chapter 5. Departure Time Choice Model

The TRANSIMS Linux version (i.e., v.3.0.2) had a feedback module of Activity Regenerator. The two primary purposes of this module include: 1) Activity Generator-to-Activity Generator feedback to clean up and reassign activities that cannot be assigned or completed by the Activity Generator, and 2) Router-to-Activity Generator feedback to solve a problem associated with activities that the Router cannot assign paths. However, there was no module for an activity rescheduling process in response to network conditions.

The TRANSIMS environment has significantly changed over the past few years. Although FHWA and open source community members continue working to improve TRANSIMS functions, the current TRANSIMS (v.4.0.1) still does not have a module for activity rescheduling, including departure time choice model, in response to network conditions. The activity rescheduling process in response to congestion/toll based on updated link travel times is important for transportation planning studies particularly in metropolitan area where significant congestion occurs in the peak period. Moreover, travelers would adjust their departure times to avoid delay or tolls. In this regard, this work is very important to better understand the travelers' behaviors in response to congestion/toll in an application of HOT lane value pricing.

The objective of this study is to extend the previous work and to further improve TRANSIMS functions. The travelers' decision on activity rescheduling behavior in response to congestion/toll is represented in the context of departure time choice behavior in this study in order to achieve a better understanding of how travelers respond to tolls/congestion in terms of route choice and departure time choice. For this purpose, a new program for the application of departure time choice model to TRANSIMS was developed (see Appendix C for source codes). It is a post-processing of the Router and the Microsimulator and represents a sequential decision making process of travelers. It utilizes travel time information feedback from both the Router and the Microsimulator. In addition to those travel times (i.e., schedule delay), individual socio-economic attributes and activity characteristics play a significant role in the decision of changing the departure time. Testing the developed program for the application of the HOT lane value pricing using the real-world data as a case study is another objective of this study. Since the Portland activity data was available, the developed program was tested using the data. The HOT lane was configured on a segment of I-5 northbound from Hwy 217 to I-405 near the CBD in Portland metropolitan region, Oregon.

The next section illustrates the simulation methodology that describes the departure time choice model. Then, simulation data and alternative scenarios are discussed, followed by simulation results.

5.1 Simulation Methodology

Figure 5.1 shows the overall conceptual models of route and departure time choices. There are two major loops, consisting of route choice loop (dotted box and lines) and departure time choice loop (thick box and lines). The route choice loop is processed by iteratively redistributing a subset of travelers between the Router and the Microsimulator. For the application of a HOT lane value pricing system, the modified Router discussed in Chapter 4 was utilized. A non-linear individual VOT takes into account three socio-economic variables (age, employment status and household income class) and one trip characteristic (trip purpose). Thus, the Router computes individual VOT for each traveler, which is combined with time-dependent tolls, computes a generalized link travel time, and finds the shortest path. The generalized link travel time function consists of two components. The first component is the 15-min time-dependent travel time obtained from the Microsimulator. The second component is the additional travel time by taking account of ratio of toll to VOT to convert monetary value to time value. Based on the plans for each traveler generated by the Router, Microsimulator simulates each vehicle second-by-second until a stabilized network is reached. Then, a subset of travelers is selected to be re-routed. The selection is based on the “travel time difference method” as discussed in Chapter 4. A stopping criterion for the route choice loop is the smallest summation of Experienced Travel Time (*ET*) over all travelers referred to as Y^*. This stopping criterion was already discussed in Chapter 4. Once a stabilized network is reached through the route choice loop, the first iteration of departure time choice is conducted, that takes into consideration the output from the route choice loop. The departure time choice model determines whether a traveler changes the departure time through 6 steps. Once a new, revised activity file is obtained from these steps, the second iteration of the route choice loop is processed, because travelers, who change their departure times, may also change their routes. Accordingly, the two loops (route choice and departure time choice loops) repeat until both stopping criteria of the two loops are met. A stopping criterion for the departure time choice loop is discussed later in this chapter.

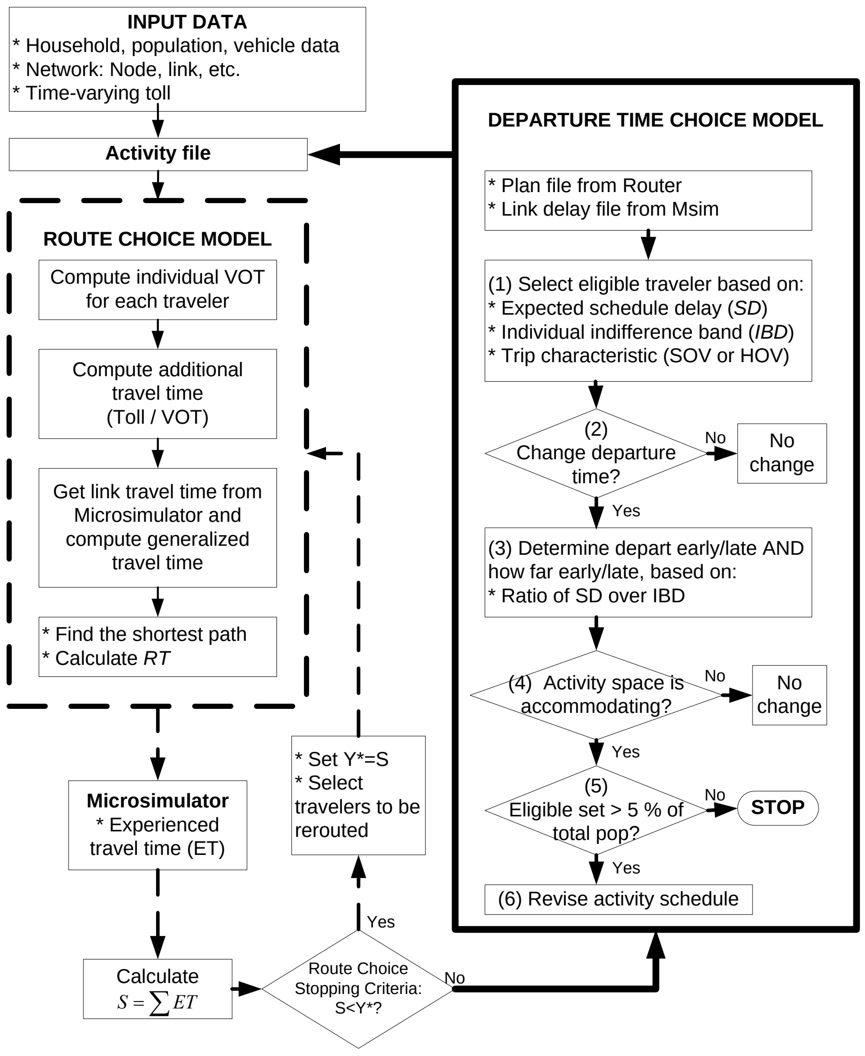

INPUT DATA
* Household, population, vehicle data
* Network: Node, link, etc.
* Time-varying toll
Activity file
ROUTE CHOICE MODEL
Compute individual VOT for each traveler
Compute additional travel time (Toll / VOT)
Get link travel time from Microsimulator and compute generalized travel time
* Find the shortest path
* Calculate RT
Microsimulator
* Experienced travel time (ET)
Calculate $S = \sum ET$
Route Choice Stopping Criteria: S<Y*?
Yes
No
* Set Y*=S
* Select travelers to be rerouted
DEPARTURE TIME CHOICE MODEL
* Plan file from Router
* Link delay file from Msim
(1) Select eligible traveler based on:
* Expected schedule delay (SD)
* Individual indifference band (IBD)
* Trip characteristic (SOV or HOV)
(2) Change departure time?
No
No change
Yes
(3) Determine depart early/late AND how far early/late, based on:
* Ratio of SD over IBD
(4) Activity space is accommodating?
No
No change
Yes
(5) Eligible set > 5 % of total pop?
No
STOP
Yes
(6) Revise activity schedule

Figure 5.1 Conceptual model.

The departure time choice model was developed as a sequential decision making process that travelers typically confront. For example, a traveler would first consider a schedule delay, which is shown in Step 1 in Figure 5.1. If there is a significant delay (i.e., arriving late to work) due to congestion and if this schedule delay is not tolerable to the traveler (Step 1), the traveler would decide whether he/she would adjust the departure time of the trip (Step 2). Once the traveler decides to change the departure time, he/she will decide the timing of departure time (i.e., 5, 10, or 15 min depart early/late than usual) (Step 3). The traveler would also check whether his/her activity schedule can accommodate the departure time changes (Step 4). The program checks the stopping criteria (Step 5) and revises the activity schedule (Step 6). The detail of this sequential decision making process of departure time choice model is discussed in the following section.

5.2 Departure Time Choice Model Procedures

Step-1: Selecting Eligible Travelers

The first step of the departure time choice model is to choose travelers eligible for changing their departure times. The eligibility of traveler is dependent on his/her activity schedules. For example, travelers who are sharing rides are excluded from the eligible travelers list because those travelers will have less flexibility in changing their departure times due to commitments such as driving children to school, car-sharing, etc. Second, the eligible travelers are chosen based on the expected schedule delay (*SD*) and individual indifference band of tolerable schedule delay (*IBD*). *SD* is the travel time difference between the Routed Travel Time (*RT*) obtained by the Router and the Experienced Travel Time (*ET*) computed by the Microsimulator. *IBD* is the tolerance level of schedule delay for each traveler. Thus, if *SD* of a trip exceeds the tolerance level (*IBD*), then this trip is eligible for further consideration. The base concept of this method is the boundedly rational model developed by Mahmassani and Liu (1999). *IBD* is different from one traveler to another, depending on the initial band and traveler characteristics. For this purpose, the model proposed by Mahamassani and Liu (1999) was adopted. The specification of *IBD* for both early and late departures utilized in this application is shown in Eq. 5.1.

$$\begin{aligned} IBD_{ij} = {} & 12.258w_i + 6.71(1\text{-}w_i) + 0.1663\, w_iAGE_i + 0.1023(1\text{-}w_i)AGE_i \\ & - 1.774\, w_iGENDER_i - 1.5238(1\text{-}w_i)GENDER_i \\ & + 0.425w_i\lambda_i(\Delta TR_{ij}/\Delta DT_{ij}) + 0.1435(1\text{-}w_i)\lambda_i(\Delta TR_{ij}/\Delta DT_{ij}) \end{aligned} \qquad (5.1)$$

where,

w_i = Binary indicator variable, 1 if $SD \geq 0$ (early schedule delay) for traveler i, and 0 otherwise (late schedule delay)

AGE_i = Age of traveler i, 1 if age<20; 2 if $20 \leq$ age ≤ 39; 3 if $40 \leq$ age ≤ 59; 4 if age ≥ 60

$GENDER_i$ = Gender of traveler i, 1 if male; 0 if female

λ_i = Binary indicator variable, 0 if $DT_{ij-1} = DT_{ij-2}$; and 1 otherwise

$\Delta TR_{ij} = ET_{ij} - ET_{ij-1}$ = Difference between travel times of traveler i in iteration j and those in iteration $j\text{-}1$ (min)

ΔDT_{ij} = Amount of departure time that traveler i has adjusted between iteration $j\text{-}1$ and $j\text{-}2$ (min)

i = Individual traveler i

j = Simulation number j (j indicates the current iteration and $j\text{-}1$ indicates the previous iteration)

In this study, the two variables (over-estimation and under-estimation errors provided by real-time information) available from the original model were excluded, since the effect of real-time information is beyond the scope of this study. The description of model estimation and verification are outside the scope of this study and readers are referred to Mahmassani and Liu (1999) for additional information.

Step-2: Departure Time Choice Utility Model

Once eligibility is determined, the next step is to decide whether the traveler is willing to change his/her departure time. For this purpose, this study adopted the Burris and Pendyala's (2002) binary logit model. Although there are good and well-known utility models related to departure time choice, most of them do not meet the purposes of this research. The proposed sequential process of departure time choice method in this study considers the Router and Microsimulator travel times, schedule delay, and *IBD* in Step 1. The level of adjustment is decided in Step 3. Accordingly, the utility function that is interested in is a binomial choice model (i.e., changing departure time or not), non-joint model, non-rescheduling model, and a utility function including individual attributes as explanatory variables. After conducting comprehensive literature reviews, only two candidate models that meet the purposes of this study were found. Between the two models, the Burris and Pendyala's model was adopted in this study. The specific reasons for the selection are discussed in section 5.3.

The utility function (U_i) used in this study for a traveler *i* willing to change his/her departure time is shown in Eq. 5.2. This case is referred to as the change departure time case. The utility of the base alternative (e.g., do not change departure time) is set to zero. Thus, a traveler who has higher probability of changing departure time of a trip than that of base utility will result in a change in his/her departure time.

$$U_i = -0.66+0.71Flexibility_i+1.1Age_i+0.93FlextimeWork_i-0.74Income_i-0.48TripPurpose_i \quad (5.2)$$

where,

$Flexibility_i$ = Flexibility in time of travel, 1 if flexibility for a traveler *i* is available, 0 otherwise

Age_i = Age group, 1 if age > 65, 0 otherwise

$FlextimeWork_i$ = flextime availability at work, 1 if a traveler *i* has a flextime at work

$Income_i$ = household income, 1 if household income > \$75,000, 0 otherwise

$TripPurpose_i$ = trip purpose, 1 for work trip, 0 otherwise

Flexibility and *FlextimeWork* variables are currently unavailable in TRANSIMS input data. Accordingly, random numbers were generated with an assumption of a certain proportion of travelers to have the two data over all populations. Note that this study used an age variable, which is available as a TRANSIMS input data, to replace the retired traveler variable in the original model. This substitution is agreeable to Burris and

Pendyala (2002) since they state that "*due to the high correlation between respondents over the age of 65 and retirees, these two variables were not used concurrently in the model.*" The description of model estimation and verification are outside the scope of this study and readers are referred to Burris and Pendyala (2002) for additional information.

Step-3: Determine Depart Early/Late and How Far Early/Late

The third step is to determine whether a traveler would depart earlier or later than usual and how far in time he/she would change the departure time. The value of SD_i is used to determine whether the traveler departs earlier or later than usual. If SD_i is positive, which means that the traveler arrives late due to congestion/toll, then he/she would shift departure time to earlier than usual (put "-" sign in Eq. 5.3). If SD_i is negative, he/she would shift departure time to later than usual (put "+" sign in Eq. 5.3).

$$\Delta t_i = (\pm)5 \text{ x } \{\text{Round of } (SD_i/IBD_i)\} \tag{5.3}$$

The decision on the levels of early/late departure time change (Δt) is based on the ratio of SD_i over IBD_i. The level of Δt is confined to a 5 minutes discrete time interval (e.g., 5, 10, 15 min, etc.). Although time is a continuous variable, an assumption of a discrete time periods with a 5 minutes time interval was made to approximate human behavior. This is consistent with typical travelers' decision because they usually report their adjusted departure times as a finite discrete time periods (Daly, et al. 2005; Small 1982).

Step-4: Checking Traveler Activity Schedule Space

In this step, the program checks the activity space of a traveler to identify whether he/she has enough leeway in his/her activity schedule to allow for a departure time change. If the adjusted departure time is conflicted with previous activity schedule, a traveler can not change the departure time of the trip. The activity space refers to the difference between the end time of activity *k-1* and the start time of activity *k* that is under consideration for rescheduling. If it is greater than an absolute value of Δt, then this person's trip *k* is eligible and his/her activity is adjusted in the next step.

Step-5 and 6: Changing Departure Time in Activity File and Stopping Criteria

Following the above steps, once the trip of traveler *i* is determined to have a change in departure time, the program adjusts traveler's activity schedule in the activity file to have a new departure time for this trip. There is no one explicit stopping criteria for the departure time choice model. As a simple approach, a number of eligible set of households was used as a stopping criterion. In other words, if the number of eligible set (the number households that change their departure times) through step 5 is less than 5% of the total population, then the program stops.

5.3 Selecting Appropriate Departure Time Choice Utility Model

As discussed in Chapter 3, many different utility models are available with different types of functions and different study purposes. The utility models associated with departure time choice can be categorized as discrete choice models (i.e., time of day models; choice models based on time intervals or on choices of departing earlier/later/no change; binomial logit model; combined model with route/mode choices), continuous models and activity scheduling/re-scheduling models.

As discussed in the previous section, the choice of timing of departure time in the proposed method will be decided by the ratio of *SD* over *IBD* in terms of 5-min intervals (refer to Step 3). Accordingly, an alternative choice set is a critical issue in selecting an appropriate utility function for this study. In this regard, most time of day models are not suitable because they are typically developed for the application of traditional 4-step TDF model and typical choice sets are peak or non-peak periods. For example, Cambridge Systematics (2005) developed a time of day model for both trip based and tour based model (Abou Zeid, et al., 2006). Although their trip based model considers both transportation attributes and individual attributes, arrival times were modeled for trips from home, while departure times were modeled for trips to home and for non-home based trips. In addition, the alternative choice sets are 30-min time intervals at best, which is still wide range for the purpose of this study. Most of multinomial logit models are not suitable either, since choice sets are still wide-range of time intervals (i.e., 30-min, 1-hr, or 2-hr).

For example, the multinomial logit choice model developed by Small (1992) is one of well-known models. The utility function includes good indicators of departure time choice (such as travel time, schedule delay, household structure, carpool, arrival flexibility, occupation type). However, the model is inclusive of work trips only. In addition, the choice sets (5-min time intervals) are based on arrival time at work, rather than departure time from home. Some multinomial logit models have choice sets of departing earlier, later, or no change. However, these types of models are also not appropriate for this research because departing earlier or later will be decided by the *SD* (i.e., whether the trip is on schedule not) in the proposed method. More recently, continuous models of departure time choice were developed because time is intrinsically a continuous variable and the utility function should theoretically be continuous as a function of time. However, the dominant model of departure time choice in practice and academic research is still a discrete choice model. In addition, a traveler would typically adjust his/her departure time as a finite discrete time periods (i.e., depart 5 or 10 minutes earlier than usual) rather than a continuous time scale (i.e., depart 4 minutes 35 seconds earlier than usual). Accordingly, a discrete time scale more precisely represents a human behavior. For this reason, continuous models were excluded in choosing an appropriate utility function. In summary, the most favorable utility function for this research is:

- A utility function that decides whether a traveler is changing (switching) departure time or not; that is, a binomial utility function will be the best option.
- Non-joint model with route and/or mode choice. This is because route choice and departure time choice models are conducted independently in TRANSIMS.
- Applicable to all types of trip purposes.
- Individual attributes. Many researchers agree that the factors of changing departure times are not only transportation system performance (i.e., travel time) but also individual's attributes (i.e., socio-economic characteristics, work conditions, flexible work time, etc.).
- Non-rescheduling models. Although an activity (tour-based) rescheduling model is good practice for the TRANSIMS application, it is out of scope this research. This would be an interesting topic for a future research.

Although there are good and well-known utility models related to departure time choice, most of them do not meet the criteria above. After conducting comprehensive literature reviews, only two candidate models were found. Mahmassani et al. (1990) developed a binomial logit model of departure time switching for AM and PM commute trips. The variables used in the utility model are travel time, lateness tolerance at work, preferred arrival time, job type, use of information. Burris and Pendyala (2002) developed a discrete choice model to examine whether a traveler would change time of travel in response to variable pricing and the frequency of change. The variables include trip purpose, flexibility in time of travel, retirement status (this is corresponding to age), flextime availability at work, and household income (> $75,000). This research adopted the Burris and Pendyala's model for the following reasons. Their model was estimated based on travel survey data (revealed preference survey) collected in variable pricing system in the Lee County, Florida. Their model includes more socio-economic variables than Mahmassani et al's model that takes only one variable (i.e., a type of job). The model of Mahmassani et al. is limited to work trip only. A travel time variable, which is included in the model of Mahmassani et al., is a good indicator of changing departure time. However, the travel time in addition to the schedule delay is already considered in the proposed method (Step 1). For these reasons, this study chose the Burris and Pendyala's model as a default model of the utility function.

5.4 Experimental Analysis Using Portland Activity Data and Network

5.4.1 Data and simulation scenarios

Data

The proposed method was tested using data from Portland, Oregon. The test network is the same as the application of HOT lane value pricing that is discussed in Chapter 4. The original Portland activity data, based on a 1996 survey data, was extracted to obtain the demand. The original TRANSIMS activity data was refined to simulate activities starting at midnight till 10 a.m. in order to include the morning peak period. Some data were suppressed. For example, activities involving transit trips were excluded because they were outside the study scope. The final testing data included a total of 120,080 households, 240,982 travelers (average 2 travelers/household), and 503,801 activities (average 2.09 activities/traveler).

Simulation Scenarios

The proposed departure time choice model was coded into TRANSIMS. The simulations were conducted for the base case and the two alternative cases. The base scenario simulates the base network, including three GPLs on I-5 northbound, but without the managed-lane system and departure time choice model. Alternative-1 scenario is the application of the managed-lane system without departure time choice model, while the departure time choice model along with the managed-lane system were tested in Alternative-2 scenario. The departure time choice model is applied to all travelers, including the managed-lane system users. The managed-lane system includes 1 HOT lane and 2 GPLs and operates with a toll schedule opening to the northbound in the morning-peak for two hours from 6 a.m. to 8 a.m. This study applied 15-min time-dependent tolls that reflect the peak congestion. Time-dependent tolls were structured by changing the toll by a value of $1.00 each 15-minute time interval (i.e., $1 during 6:00-6:15 a.m., $2 during 6:15-6:30 a.m., $3 during 6:30-6:45 a.m., $4 during 6:45-7:00 a.m., $4 during 7:00-7:15 a.m., $3 during 7:15-7:30 a.m., $2 during 7:30-7:45 a.m., and $1 during 7:45-8:00 a.m.). Note that this case study did not implement the 15-min dynamic tolls, because the output of dynamic tolls presented in Chapter 4 was not available at the time of this study.

5.4.2 Experimental results

Each scenario involved multiple simulation runs. In all cases, about 9 iterations were conducted for Router/Microsimulator stabilization. For comparison purposes, the same control parameters were used in all cases. This section presents the analyses of the effects of the managed-lane system and departure time choice on travelers' behavior.

Table 5.1 summarizes overall results. In Alternative-2 scenario, after the first departure time choice (DTC) iteration, 12,061 households (10% out of overall population) changed their departure times. After the second DTC iteration, additional 4,634 (3.9%)

households changed their departure times. The DTC iteration was stopped after the second iteration, because the percentage of households, who changed their departure times, was decreased to 3.9%, which is less than the 5% stopping criteria. The table also shows the number of rescheduled activity records by households that changed their departure times (16,669 and 5,966 records in the two iterations, respectively). Table 5.1 also presents the overall network performance, including vehicle-miles traveled (VMT), vehicle-hours traveled (VHT) and hours of delay (HOD) that are collected from the overall Portland network. As a result of the DTC model, overall network was improved. Compared to Alternative-1 scenario, VHT and HOD was decreased by 2.78% and 8.20%, respectively after the second DTC iteration. However, VMT was slightly increased by 0.07%. The increased VMT and the decreased VHT imply that some travelers detour their trips, but with lower total travel times. For example, travelers, who change their departure times, can also change their shortest paths through the route choice loop. The new paths may lead to longer trip lengths, but they provide lower travel times than before. The base scenario output presented in Table 5.1 is for reference purpose only because the network has changed in Alternatives 1 and 2.

Table 5.1 Comparison of All Scenarios

		Base	Alternative 1 (managed-lane system)	Alternative 2 (managed-lane system + DTC model)	
				1st DTC run	2nd DTC run
DTC application	Number of households that changed departure time	N/A	N/A	12,061 (10%)	4,634 (3.9%)
	Number of rescheduled activity records	N/A	N/A	16,669	5,966
Overall network performance	Veh-miles traveled	1,130,020	1,130,924	1,132,106	1,131,737
	Veh-hours traveled	48,644	48,683	47,429	47,331
	Hours of delay	19,883	19,899	18,362	18,267

In this unique managed-lane system, traffic volumes in the HOT lane are decreasing on the links approaching downtown because there is only one entering point, and some travelers are leaving the HOT lane through the provided exits. On the other hand, traffic volumes in the GPLs are increasing on the links toward downtown because additional travelers are entering the GPLs in the middle of the system. The comparison of the hourly volumes per lane counted at 4 different locations in the system in the Alternative 1 case is shown in Figure 5.2. Note that location # 1 in this figure indicates the first link of the managed-lane system and location # 4 indicates the ending link of the system, while two other locations are links in the middle of the system. As expected, the hourly per-lane volumes in the GPLs are increasing as moving toward downtown, while those in the HOT lane are decreasing as moving toward downtown. Significant congestion was observed in the GPLs than in the HOT lane.

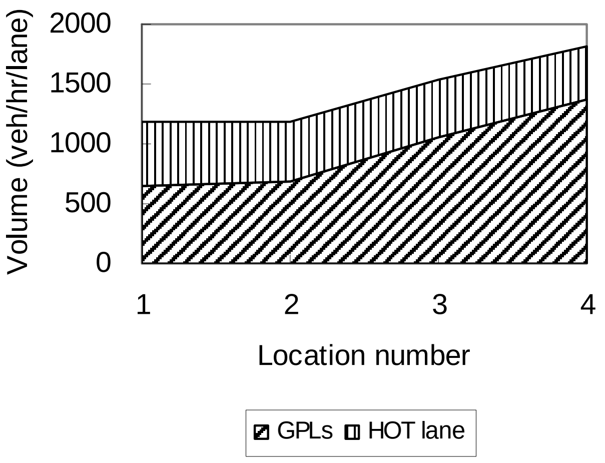


Figure 5.2 Comparison of hourly per-lane volume in selected 4 locations.

Table 5.2 presents the comparison of link volumes (veh/hr/lane) at a selected location by 15-min time of day between the two scenarios. The selected location A is close to the CBD in the managed-lane system, and is shown in Figure 4.15. As expected, the table illustrates the effect of departure time changes and/or route changes. Compared to the

Alternative-1, traffic volumes of the GPLs and the HOT lane respectively were decreased in Alternative-2 in most of 15-min time periods.

Table 5.2 Comparison of Link Volumes (veh/hr/lane)

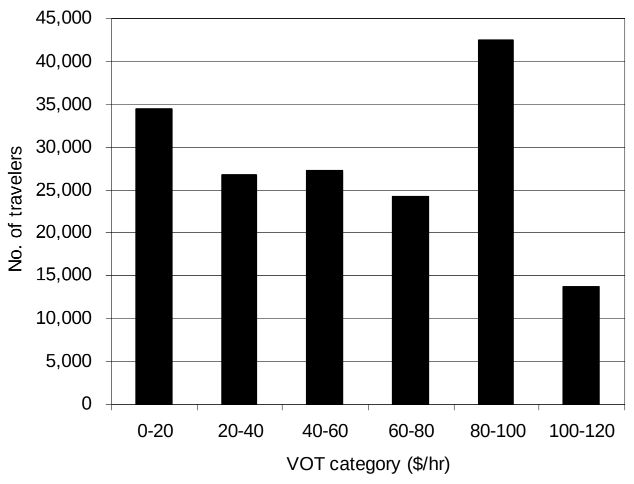
Time (a.m.) and Scenario	6:00-6:15	6:15-6:30	6:30-6:45	6:45-7:00	7:00-7:15	7:15-7:30	7:30-7:45	7:45-8:00
Alternative-1 (managed-lane system)	1,620	1,812	1,456	1,836	1,948	1,984	1,272	1,187
Alternative-2 (managed-lane system + DTC model)	1,520	1,693	1,507	1,631	1,800	1,717	1,223	1,137

One of unique benefits of using TRANSIMS is that it traces not only vehicle movement, but also individual traveler movement, which include socio-economic and trip characteristics that are identifiable. Accordingly, it allows an analyst to identify and trace each traveler, who changed his/her departure time, the traveler socio-economic characteristics, and which route he/she has chosen. For this purpose, some travelers were randomly selected who changed their departure times. Table 5.3 presents their *SD*, *IBD*, Δt, and individual characteristics. For example, a traveler (HH id 379 and person id 1) was a HOT lane user in Alternative-1 scenario, and she is is 36 years old, has a flexible work time. As a result of higher *SD* (about 13 minutes) than *IBD* (about 4 minutes), she changed her departure time by 15 minutes earlier to go to work after the first iteration of the DTC model.

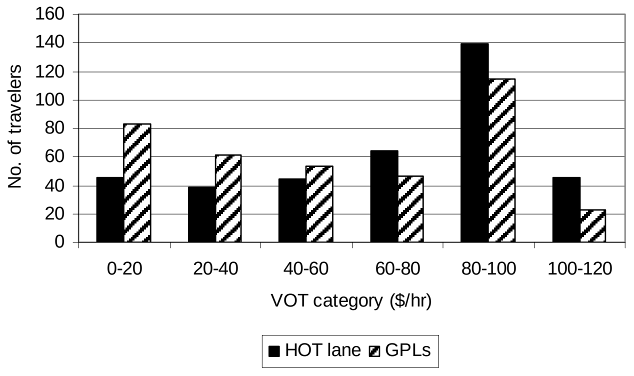
Table 5.3 Selected Travelers who Changed Their Departure Times

HH	Person	RT (min)	ET (min)	SD (min)	IBD (min)	dT (min)	Age	Gender (1-male, 0-female)	Flexibility (1-yes, 0-no)	Flextimework (1-yes, 0-no)	Trip purpose (1-work, 0-nonwork)
379	1	11	24	13	4	-15	36	0	1	1	1
815	1	10	21	11	4	-15	74	1	1	1	1
2936	3	15	25	10	5	-10	17	0	1	1	1
2936	1	15	26	11	5	-10	42	0	1	1	1
9579	2	10	21	11	6	-10	40	1	1	1	0
13899	1	9	19	10	5	-10	32	0	1	1	1
15760	2	15	25	10	5	-10	31	1	1	1	1
16036	4	5	15	10	5	-10	67	1	1	1	1
16327	1	6	17	11	6	-10	37	0	1	1	1
17502	4	10	20	10	5	-10	18	1	1	1	1
17860	2	4	15	11	5	-10	42	1	1	1	1
17875	2	14	25	11	5	-10	26	1	1	1	1
18400	2	14	25	10	5	-10	28	1	1	1	1
18485	2	10	20	11	5	-10	33	1	1	1	1
18952	1	7	20	12	6	-10	41	0	1	1	1
19937	2	12	26	14	7	-10	36	0	1	1	0
21737	2	14	25	12	4	-15	28	1	1	1	0
23324	2	13	25	12	4	-15	55	1	1	1	0
23903	1	9	22	13	4	-15	53	0	1	1	0
23939	2	16	26	10	3	-15	25	1	1	1	1

Other results that illustrate the effect of individual VOT and a toll on route choice during 7:00-7:15 a.m. in Altenative-1 scenario are shown in Figure 5.3. Figure 5.3(a) shows the number of total travelers in overall network classified by VOT. The values are skewed to the right because travelers commuting from the suburbs to downtown during peak periods represent high-income households. Figure 5.3(b) shows the number of travelers classified by VOT at the first link of the managed-lane system, who are choice users between the HOT lane and the GPLs. As expected, higher percentages of travelers have chosen the HOT lane in higher VOT categories. In lower VOT categories, more travelers have chosen the GPLs. A similar result was also obtained in Alternative-2 scenario. It should be noted that choosing between the HOT lane and the GPLs depends not only on VOT but also on trip characteristics (i.e., OD of the trip, trip length). For example, travelers with high VOTs may choose the GPLs if their trip lengths are short and they have to exit early in the middle of the system. Travelers with low VOTs may choose the HOT lane if their trip lengths are long.



(a)



(b)

Figure 5.3 Number of travelers by 6 VOT groups: (a) in all population and (b) in the start links of the managed-lane system during 7:00-7:15 a.m. (Alternative-1)

Chapter 6. Summary and Conclusions

Value pricing is an economic concept that drivers pay for driving on a certain roadway or entering into a particular area. It is now an accepted strategy for congestion and demand management in metropolitan areas. The travelers have some degree of choice over whether they travel on the toll lane by paying tolls or on the non-toll lanes without charges in a value pricing system. The most prominent benefit of value pricing is that it can encourage some trips during peak periods to shift to off-peak time, less congested routes, alternative modes, or shared driving. Along with alternate congestion management strategies, many transportation agencies have started looking at value pricing as a method to help financial shortfalls of new congestion management projects. Value pricing allows revenue collected from toll facilities to reduce operational concerns with underutilized High Occupancy Vehicle (HOV) facilities and relieves environmental concerns. The advancements in electronic toll collection technologies have made the implementation of value pricing systems more realistic and attractive. Recently, transportation agencies have become increasingly interested in HOT lanes value pricing system with time-dependent tolls or dynamic tolls that change by the congestion level.

Over the last decade, the concept of road pricing has been advocated by many economists. Based on the fundamental economic principle of marginal-cost pricing, economists argue that a road user using a congested road should pay a toll equal to the difference between the marginal social cost and the marginal private cost in order to maximize social net benefit and to internalize the externality by imposing congestion toll. Several different economic, mathematical and programming approaches were proposed to solve value pricing modeling. However, the analytical approaches have their own limitations. Although approaches based on simulation models were developed and solved some issues of limitations of analytical models, the existing simulation-based models have still limitations such as the assumption of homogenous VOT. The current travel demand forecasting models cannot efficiently evaluate and determine the impacts of pricing on travel behavior due to aggregate and zonal based approaches.

This paper proposed a methodology for the analysis of HOT lane value pricing with dynamic tolls, simultaneously taking into account user heterogeneity of VOT to estimate the travelers' response to tolls in choosing their routes. The approach was designed to determine 15-min toll rates that dynamically vary with the congestion level in the HOT lane(s). The use of non-linear, individual VOT in this research relaxes the conventional assumption of constant VOT or randomly distributed VOT over the population, which was frequently used in previous studies. To achieve these objectives, the microscopic simulation model, TRANSIMS, was used, and the source codes of TRANSIMS were modified in two different versions, including the old version (available in Linux machine only) and new version (available in both Windows and Linux machines). The proposed methodology was successfully implemented in TRANSIMS and it improved the functions of TRANSIMS for the application of HOT lane value pricing. The modified Router finds the shortest path for each traveler based on the generalized travel time function that simultaneously take into account 15-min dynamic tolls and individual VOT. In addition, the modified version allows for the user to specify different coefficient values to compute VOT. Accordingly, it provides flexibility for VOT function.

The modified versions were tested and run using two different data sets, including simplified network and real-world data from Portland, OR. The first case study simulates travelers for two hours in the morning peak in the simplified network. The application results demonstrated the feasibility of the proposed simulation methodology and the sensitivity of route choice to different toll levels. The outputs from the microscopic analysis clearly indicate the effectiveness of the analysis in analyzing travelers' route choice behavior based on different socio-economic and travel characteristics when different toll rates are applied. The results were consistent with intuition; SOVs with higher VOTs are more likely to choose HOT lanes. The case study presented the comparison of dynamic tolls based on different pricing decision method and the sensitivity of route choice based on different socio-economic characteristics. In the second case study, the two scenarios (the base case without tolls and the alternative case with dynamic tolls) were run using the real-world data from Portland, OR. The managed-lane system was applied to I-5 northbound in Portland, Oregon in the alternative scenario. The results showed that veh-hours traveled (VHT) and hours of delay (HOD) in overall network were improved by 3.32 % and 7.81 %, respectively. The experimental results presented 15-min dynamic tolls and comparisons of traffic volumes in the HOT lane and in the GPLs. When comparing VOTs categorized by 6 groups between the HOT lane users and the GPLs users, the results were logical and reasonable: that is, when there is no toll, VOTs did not impact the selection of route choice between the HOT lane and the GPLs. On the other hand, when there are tolls, higher VOT travelers are more likely to choose the HOT lane. Finally, the case study presented the comparison of travel times of the selected travelers between the base case and the alternative case. It shows how much they can save their travel times by choosing the HOT lane.

In addition to the development of the dynamic value pricing along with individual VOT, the departure time choice model was also developed. The traveler's behavior in response to congestion/toll was represented in the departure time choice model in this research. Since there is currently no activity rescheduling process, including departure time choice, in response to congestion/tolls in the existing TRANSIMS version (4.0.1), the new program was developed and successfully implemented into TRANSIMS. The proposed methodology was run using Portland data. In addition to the base case (without the managed-lane system and departure time choice), two alternative scenarios were simulated with time-dependent tolls and with the application of departure time choice model. The impact of departure time choice shows an improvement in overall system-wide performance in terms of VHT and HOD. As a result of departure time choice model, 13.9% of total households did change their departure times by shifting to non-peak period. The impact of VOT on route choice was also presented. One of unique benefits of TRANSIMS is that it traces individual traveler movements as well as vehicle movements. It allows the analysts to identify individual socio-economic and trip characteristics, and to analyze impacts of various transportation policies (i.e., tolls) on the traveler's behavior. The paper presented an example of how individuals adapted their behaviors in terms of route and departure time choices.

TRANSIMS is an activity-based simulation model with unique philosophies. As the next generation of transportation planning and simulation model, the unique features of

TRANSIMS promise several extensions for the future works, including, for example, the effects of the HOT lane value pricing on the mode shifts to HOVs and vehicle-to-vehicle communication as part of advanced traveler information system. For a departure time choice model, this study applied a trip-based utility function. However, since TRANSIMS is an activity (tour) based simulation model, a (tour-based) activity rescheduling utility model can be developed and applied to TRANSIMS, which is an interesting topic for future research. The proposed methodology utilized deterministic functions (i.e., VOT function). However, some components (or variables) can be treated in a randomly or stochastic way. This would be also an interesting research for the future work.

References

Abdelghany, A.F., K.F. Abdelghany, H.S. Mahmassani, and P.M. Murray. Dynamic traffic assignment in design and evaluation of High-Occupancy Toll Lanes. In Transportation Research Record 1733, TRB, National Research Council, Washington, D.C., pp. 39-48. 2000.

Abou-Zeid, M, and I. Chabini. Methods for congestion pricing in dynamic traffic networks, Accepted at the 10th IFAC Symposium on Control in Transportation Systems, August 4-6, Seikei Univ., Tokyo, Japan. 2003.

Abou Zeid, M., T. Rossi, and B. Gardner. Modeling time-of-day choice in context of tour- and activity-based models. Transportation Research Record: Journal of the Transportation Research Board, No.1981.pp.42-49. 2006.

Agnew, C.E. The theory of congestion tolls, Journal of Regional Science, Vol. 17, pp.381-393. 1977.

Algers, S., P. Bergstrom, M. Dahlberg, and J.L. Dillen. Mixed logit estimation of the value of travel time. Working Paper, Department of Economics, Uppasala University, Sweden. 1998.

Amador, F.J, R.M. Gonzalex, and J.D. Ortuzar. Preference heterogeneity and willingness to pay for travel time. Facultad de Ciencias Economicas de la ULPGC. 2004. Available:http://www.bibliotecas.ulpgc.es/fcee/hemeroteca/documentos%20de%20trabajo/DocumentosDTrabajo/doc52/DT2004-12.pdf, Mar., 2005.

Armstrong, P., R. Garrido, and J. Ortuzar. Confidence intervals to bound the value of time. Transportation Research E, Vol. 37, pp.143–161. 2001.

Arnott, R., A. de Palma, and R. Lindsey. Departure time and route choice for the morning commute. Transportation Research B, Vol. 24, No. 3, pp.209-228. 1990.

Arnott, R., A. de Palma, and R. Lindsey. Structural model of peak-period congestion: a traffic bottleneck with elastic demand. The American Economic Review, Vol. 24, pp.209-228. 1993.

Bard, J.F. Practical Bilevel Optimization: Algorithms and Applications. Kluwer Academic Publishers, Dordrecht/Boston/London. 1998.

Ben-Akiva, M., A. de Palma, and P. Kanaroglou. Dynamic model of peak traffic congestion with elastic arrival rates, Transportation Science, Vol. 20, pp.164-181. 1986

Ben-Akiva, M and S.R. Lerman. Discrete choice analysis: theory and application to travel demand. The MIT Press. 1985.

Bhat, C.R. and J.L. Steed. A continuous-time model of departure time choice for urban shopping trips, Transportation Research Part B, Vol. 36, pp.207-224. 2002.

Braid, R.M. Peak-load pricing of a transportation route with an unpriced substitute. Journal of Urban Economics, Vol. 40, pp. 179-197. 1996.

Brownstone, D., A. Ghosh, T.F. Golob, C. Kazimi, and D. van Amelsfort. Drivers' willingness-to-pay to reduce travel time: evidence from the San Diego I-15 congestion pricing project. Transportation Research, Part A, vol. 37, pp.372-387. 2003.

Brownstone, D. and K.A. Small. Valuing time and reliability: assessing the evidence from road pricing demonstrations, Transportation Research, Part A. Vol. 39, pp.279–293. 2005.

Burris, M.W. Lee County variable pricing project: evaluation report. Center for Urban Transportation Research, College of Engineering, University of South Florida. 2001.

Burris, M.W. and R.L. Hannay. Equity analysis of the Houston Quickride project. The 82nd Annual Meeting of the Transportation Research Board, Washington, DC. 2003.

Burris, M.W. and R.M. Pendyala. Discrete choice models of traveler participation in differential time of day pricing programs, Transport Policy, Vol.9, No.3, pp.241-251. 2002.

Burris, M.W. and C.R. Swenson. Survey of Lee County variable pricing project participants. Summer Meeting of the Transportation Research Board, Vail, CO. 2001.

Cambridge Systematics. Forecasting Person Travel by Time of Day. FHWA, U.S. Department of Transportation, 2005.

Carey, M., and A. Srinivasan. Externalities, average and marginal costs, and tolls on congested networks with time-varying flows. Operations Research, Vol. 41, pp.217-231. 1993.

Chang, S. and C. Cheng. A time-varying congestion pricing model for optimization of transportation corridor. Presented at the 82nd annual meeting of the Transportation Research Board, Washington, D.C. 2003.

Chen, H.K. Dynamic Travel Choice Models. A Variational Inequality Approach. Springer, New York. 1999.

Chen, M., and D.H. Bernstein. Solving the toll design problem with multiple user groups. Transportation Research B, Vol. 38, pp.61-79. 2004.

Cherchi, E. and J. de D. Ortuzar. Alternative specific variables in non-linear utility functions: influence of correlation, homoscedasticity and taste variations. Presented at the 10th International Conference on Travel Behavior Research, Lucerne. 2003.

Dafermos, S. Traffic equilibrium and variational inequalities, Transportation Science, Vol. 14, pp.42-54. 1980.

Daly, A., S. Hess, J.W. Polak, G. Hyman, and C. Rohr. Modelling departure time and mode choice, ERSA conference paper, European Regional Science Association. 2005.

de Palma, A., M. Kilani, and R. Lindsey. Congestion pricing on a road network: A study using the dynamic equilibrium simulator METROPOLIS. Transportation Research part A, Vol. 39, pp. 588-611. 2005.

de Palma, A. and R. Lindsey. Private toll roads: A dynamic equilibrium analysis, Proceedings of the International Conference: Rehabilitation and Development of Civil Engineering Infrastructure Systems, Beirut, Lebanon, June 9-11. 1997.

de Palma, A. and R. Lindsey. Private toll roads: Competition under various ownership regimes, Annals of Regional Science, Vol. 34, pp.13-35. 2000.

de Palma, A., and R. Lindsey. Congestion pricing with heterogeneous travelers: a general-equilibrium welfare analysis, Networks and Spatial Economics, Vol.4, pp.135-160. 2004.

Ettema, D. and H. Timmermans. Modeling departure time choice in the context of activity scheduling behavior. Transportation Research Record: Journal of the Transportation Research Board, No.1831., pp.39-46. 2003.

Fisk, C. and D.E. Boyce. Alternative variational inequality formulations of network equilibrium travel choice problem, Transportation Science, Vol. 17, pp.454-463. 1983.

Fosgerau, M. Investigating the distribution of the value of travel time savings. Economics Working Paper, Urban/Regional, 0410005. 2004.
Available: http://econwpa.wustl.edu:80/eps/urb/papers/0411/0411006.pdf, Mar., 2005.

Freisz T.L., D. Bernstein, T.E. Smith, R.L. Tobin, and B.W. Wie. A Variational Inequality formulation of the dynamic network user equilibrium problem. Operations Research, Vol. 41, pp.179-191. 1993.

Friesz,T.L., J. Luque, R. Tobin, and B.W. Wie. Dynamic network traffic assignment considered as a continuous time optimal control problem, Operations Research, Vol. 37, pp.893-901. 1989.

Giuliano, G. "Equity and fairness considerations of congestion pricing," in Curbing Gridlock, TRB, National Academy Press, Washington D.C., p.p. 250-279, 1994.

Ghosh, A. Heterogeneity in value-of-time: revealed and stated preference estimates from the I-15 congestion pricing project. Working Paper. Department of Economics, University of California, Irvine, CA. 2000.

Hendrickson, C. and E. Plank. The flexibility of departure times for work trips. Transportation Research Part A, Vol. 18, pp.25-36. 1984.

Hensher, D.A., and P. Goodwin. Using values of travel time savings for toll roads: avoiding some common errors. Transport Policy, Vol. 11, pp.171–181. 2004.

Hubert H. Humphrey, Value Pricing Homepage, Hubert H. Humphrey Institute of Public Affairs, University of Minnesota.
Available: http://www.hhh.umn.edu/centers/slp/vp/vp_org/, Feb., 2009.

I-15 FasTrak Program. Available: http://fastrak.sandag.org/, Feb., 2009.

Jeihani, M., H. Sherali, and A. Hobeika. Computing dynamic user equilibria for large-scale transportation networks. Transportation, 33(6), pp.589-604. 2006.

Joh, C-H, S.T. Doherty, and J.W. Polak. Analysis of factors affecting the frequency and type of activity schedule modification. Transportation Research Record: Journal of the Transportation Research Board, No.1926, pp.19-25. 2005.

Johnston, R., J. Lund, and P. Craig. Capacity-allocation methods for reducing urban traffic congestion. Journal of Transportation Engineering, Vol. 121, No. 1, pp.27-39. 1995.

Kain, J. “Impacts of congestion pricing on transit and carpool demand and supply,” in Curbing Gridlock, TRB, National Academy Press. Washington D.C., pp.502-553. 1994.

Konig, A., G. Abay, and K.W. Axhausen. Time is money: the valuation of travel time savings in Switzerland, Presented at the 3rd Swiss Transport Research Conference, Ascona. 2003.

Kwon, E., C. Kelen, B. Ran, R. He. Hierarchical evaluation of HOT Lane operations using dynamic network models. Presented at the 79th Annual Meeting of the Transportation Research Board, Washington, D.C., 2000.

Lam, T. Evaluating value-pricing projects with both scheduling and route choices. Regional Science and Urban Economics, Vol. 34, pp. 225-240. 2004.

Lam, W. and H. Huang. Dynamic user optimal traffic assignment model for many to one travel demand. Transportation Research B, Vol. 29, pp.243-259. 1995.

Lee, K., and A. Hobeika. Application of dynamic value pricing through enhancements to TRANSIMS. Transportation Research Record, No. 2003, Transportation Research Board, Washington, D.C., pp.7-16. 2007.

LeeWay Service Center. Available: http://www.leewayinfo.com, Feb., 2009.

Litman, T. Using road pricing revenue: Economic Efficiency and Equity Considerations, Transportation Research Record 1558, Transportation Research Board, pp.24-28, 1996.

Litman, T. Evaluating transportation equity. Victoria Transport Policy Institute, Victoria, BC, Canada. 1999.

Liu, L.N., and D.E. Boyce. Variational inequality formulation of the system-optimal travel choice problem and efficient congestion tolls for a general transportation network with multiple time periods. Regional Science and Urban Economics, Vol. 32, pp.627-650. 2002.

Liu, L.N., and J.F. McDonald. Efficient congestion tolls in the presence of unpriced congestion: a peak and off-peak simulation model. Journal of Urban Economics, Vol. 44, pp.352-356. 1998.

Liu, L.N., and J.F. McDonald. Economic efficiency of second-best congestion pricing schemes in urban highway systems. Transportation Research B, Vol. 33, pp.157-188. 1999.

London Congestion Charging.
Available: http://www.tfl.gov.uk/roadusers/congestioncharging/, Feb., 2009.

Mackie, P.J., M. Wardman, A.S. Fowkes, G. Whelan, J. Nellthorp, and J. Bates. Values of travel time savings in the UK. Report to Department for Transport. 2003.

Mahmassani, H.S., C. Caplice, and M. Walton. Characteristics of urban commuter behavior: switching propensity and use of information. Transportation Research Record No. 1285, Transportation Research Board, Washington, D.C., 57-69. 1990.

Mahmassani, H.S, and G. Chang. Dynamic aspects of departure time choice behavior in a commuting system: theoretical framework and experimental analysis. Transportation Research Record: Journal of the Transportation Research Board, No.1037, pp.88-101. 1985.

Mahmassani, H.S. and Y. Liu. Dynamics of commuting decision behavior under advanced traveler information systems, Transportation Research Part C, Vol. 7, pp.91-107. 1999.

Mahmassani, H.S., X. Zhou, and C. Lu. Toll pricing and heterogeneous users: approximation algorithms for finding bi-criterion time-dependent efficient paths in large-scale traffic networks. Presented at the 84th annual meeting of the Transportation Research Board, Washington, D.C. 2005.

Mannering, F.L. Poisson analysis of commuter flexibility in changing routes and departure times. Transportation Research Part B, Vol.23, No.1, pp.53-60. 1989.

Miller, T.R. The Value of Time and the Benefit of Time Saving: A Literature Synthesis and Recommendations on Values. The Urban Institute, Washington, D.C. 1989.

Murray, P., H.S. Mahmassani, A. Abdelghany, and S. Handy. Defining special-use lanes: case studies and guidelines. Center for Transportation Research, University of Texas, Austin, 2000.

Nagurney, A. Network Economics: A Variational Inequality Approach, Kluwer Academic Publishers, Massachusetts. 1993.

National Research Council. Curbing Gridlock; Peak-Period Fees to Relieve Traffic Congestion. Special Report #242. Washington, DC: National Academy Pres. 1994.

NCHRP 08-57. Improved Framework and Tools for Highway Pricing Decisions. FHWA.

Ortuzar, J. D., and L.G. Willumsen. Modelling transport. Wiley, New York, NY. 2001.

Ran, B., R.W. Hall, and D.E. Boyce. A link based Variational Inequality model for dynamic departure time/route choice. Transportation Research B. Vol. 30, No.1, pp.31-46. 1996.

Romero, J., H. Morisugi, and T. Moriguchi. Short confidence intervals for value of time. Presented at the 84th annual meeting of the Transportation Research Board, Washington, D.C. 2005.

Saleh, W and S. Farrell. Implications of congestion charging for departure time choice: work and non-work schedule flexibility, Transportation Research Part A, Vol. 39, pp.773-791. 2005.

SANDAG. Report to the California legislature: San Diego's Interstate 15 congestion pricing & transit development demonstration program. San Diego Association of Governments. 1999.

SANDAG. I-15 Congestion pricing project: monitoring and evaluation services task 11, phase II Year Three enforcement effectiveness and violation assessment. San Diego Association of Governments. 2001.

Shaver, K. Virginia willing to study toll lane potential. Washington Post, January 16, B01. 2003.

Small, K. A. The scheduling of consumer activities: work trips. American Economic Review, Vol. 72, pp.467-479. 1982.

Small, K.A. Urban Transportation Economics. Hardwood Academic Publishers. 1992a.

Small, K.A. Using the revenues from congestion pricing: A Southern California Case Study. Reason Public Policy Institute. 1992b.

Small, K.A., R. Noland, X. Chu, and D. Lewis. Valuation of travel-time savings and predictability in congested conditions for highway user-cost estimation, National Cooperative Highway Research Program Report 431, Transportation Research Board, Washington D.C. 1999.

Small, K.A., C. Winston, and J. Yan. Uncovering the distribution of motorists' preferences for travel time and reliability: implications for road pricing. Accepted manuscripts in Econometrica. 2003.

Small, K. and J. Yan. The value of "value pricing" of roads: second-best pricing and product differentiation, Journal of Urban Economics, Vol.49, pp.310-336. 2001.

Smith, L. Congestion pricing. ITS Decision Report. 2002. Available: http://www.calccit.org/itsdecision/serv_and_tech/Travel_demand_man/Congestion_pricing_tdm/con_pricing_report_tdm.htm, Feb., 2009.

Smith, M.J. The existence, uniqueness and stability of traffic equilibria, Transportation Research Part B, Vol. 13, pp.295-304. 1979.

SR-91 Express lanes. Available: http://www.91expresslanes.com/, Feb., 2009.

Srinivasan, K.K. and H.S. Mahmassani. Role of congestion and information in trip-makers' dynamic decision processes: experimental investigation, Transportation Research Record 1676, pp.44-52. 1999.

Steimetz, S.C. and D. Brownstone. Estimating commuters' "value of time" with noisy data: a multiple imputation approach. Transportation Research, Part B, Vol. 39, pp.865-889. 2005.

Sullivan, E. Continuation study to evaluate the impacts of the SR 91 Value-Priced Express Lanes: final report. California Department of Transportation, Traffic Operations Program, HOV Systems Branch. 2000.

TRANSIMS Open Source. Available: http://transims-opensource.org, Oct., 2008.

Transport for London. Central London congestion charging, impacts monitoring program: preview of first annual report. Impacts Monitoring Group of Congestion Charging Division in Transport for London. 2003.

Value Pricing Pilot Program, FHWA. Available: http://www.fhwa.dot.gov/policy/vppp.htm, Feb., 2009.

Verhoef, E.T. The implementation of marginal external cost pricing in road transport, Regional Science, Vol. 79, pp.307-332. 2000.

Verhoef, E. T. Second-best congestion pricing in general static transportation networks with elastic demands. Regional Science and Urban Economics, Vol. 32, pp.281-310. 2002a.

Verhoef, E.T. Second-best congestion pricing in general networks; heuristic algorithms for finding eecond-best toll levels and toll points, Transportation Research Part B, Vol. 36, pp.707-729. 2002b.

Vickrey, W.S. Congestion theory and transport investment. American Economic Review, Vol. 59, pp.251-261. 1969.

Viti, F., S.F. Catalano, L. Minwei, C. Lindveld, and H. van Zuylen. An optimization problem with dynamic route-departure time choice and pricing. Presented at the 82nd annual meeting of the Transportation Research Board, Washington, D.C. 2003.

VTPI, Road Pricing, Victoria Transport Policy Institute. Available: http://www.vtpi.org/tdm/tdm35.htm, Feb., 2009.

Wang, J. J. Timing utility of daily activities and its impact on travel. Transportation Research Part A, Vol. 30, pp.189-206. 1996.

Wardman, M. The value of travel time: a review of British evidence. Journal of Transport Economics and Policy, Vol. 32, pp.285-316. 1998.

Whelan, G. and J. Bates. Market segmentation analysis. ITS Working Paper 565, Institute for Transport Studies, University of Leeds, 2001.

Wie, B.W. A convex control model of dynamic system optimal traffic assignment, Preprints of the 8th IFAC/IFIP/IFORS Symposium in Chania, Greece, Vol. II, pp.540-545. 1997.

Wie, B.W. A convex control model of dynamic system optimal traffic assignment, Control Engineering Practice, Vol. 6, pp.745-753. 1998.

Wie, B.W., R.L. Tobin, and T.L. Friesz. The augmented Lagrangian method for solving dynamic network traffic assignment models in discrete time. Transportation Science, Vol. 28, pp.204-220. 1994.

Wilbur Smith Associates. I-15 Managed Lanes value pricing project planning study (Volume 1: traffic, revenue, and toll operations), San Diego Association of Governments. 2002.

Yamamoto, T., S. Fujii, R. Kitamura, and H. Yoshida. Analysis of time allocation, departure time, and route choice behavior under congestion pricing, Transportation Research Record 1725, TRB, National Research Council, Washington, D.C., pp.95-101. 2000.

Yang, H. System optimum, stochastic user equilibrium, and optimal link tolls. Transportation Science, Vol. 33, pp.354–360. 1999.

Yang, H., and M. Bell. Traffic restraint, road pricing, and network equilibrium. Transportation Research B, Vol. 31, No. 4, pp.303-314. 1997.

Yang, H., and J. Huang. Analysis of the time-varying pricing of a bottleneck with elastic demand using optimal control theory, Transportation Research B, Vol. 31, No. 6, pp.425-440. 1997.

Yang, H., and J. Huang. Principle of marginal-cost pricing: How does it work in a general road network? Transportation Research A, Vol. 32, No. 1, pp.45-54. 1998.

Yang, H., and J. Huang. The multi-class, multi-criteria traffic network equilibrium and systems optimum problem, Transportation Research B, Vol. 38, No. 1, pp.1-15. 2004.

Yang, H., and W. Lam. Optimal road tolls under conditions of queuing and congestion. Transportation Research A, Vol. 30, pp.319-332. 1996.

Yang, H., and Q. Meng. Departure time, route choice and congestion toll in a queuing network with elastic demand, Transportation Research B, Vol. 32, No. 4, pp.247-260. 1998.

Yang, H., W.H. Tang, W.M. Cheung, and Q. Meng. Profitability and welfare gain of private toll roads in a network with heterogeneous users. Transportation Research A, Vol. 36. pp.537–554. 2002.

Yang, H., and X. Zhang, X. Multiclass network toll design problem with social and spatial equity constraints. Journal of Transportation Engineering, Vol. 128, pp.420-428. 2002.

Yin, Y., and H. Yang. Optimal tolls with multiclass, bicriterion traffic network equilibrium. Presented at the 83rd annual meeting of the Transportation Research Board, Washington, D.C. 2004.

Zhao, Y., and K. Kockelman. Online marginal-cost pricing across networks: Incorporating heterogeneous users and stochastic equilibria. Presented at the 83rd annual meeting of the Transportation Research Board, Washington, D.C. 2004.

Appendix

Appendix A. Summary Table on Added/Modified Source Codes for VOT and Dynamic Tolls

File directory and name	Added and/or modified function name	Purpose
/Router/Drive_Path.cpp	void Router::Output_Dtoll(int volume, int capacity, double ratio, int basetoll, int newtoll, DToll_Data* toll_ptr_cp)	Output the dtoll
/Router/Drive_Path.cpp	double Router::Base_Toll(int volume, int capacity, DToll_Data* toll_ptr_cp)	Calculate the base toll
/Router/Drive_Path.cpp	void Router::Setup_dtoll_cp_array (void)	Make a copy of dtoll array
/Router/Drive_Path.cpp	void Router::copy_dtoll_ptr (DToll_Data* toll_ptr, DToll_Data* toll_ptr_tmp)	Copy the dtoll data
/Router/Drive_Path.cpp	double Router::Cal_Dtoll(int volume, int capacity, DToll_Data* toll_ptr_cp)	Calculate the new toll
/Router/Drive_Path.cpp	double Router::Update_Dtoll_Imp(Link_Data *link_ptr,DToll_Data *toll_ptr, DToll_Data *toll_ptr_cp, TTime_Data *ttime_ptr, bool ab_flag,int dir)	Update the dynamic toll
/Router/Drive_Path.cpp	double Router::Cal_Write(Link_Data *link_ptr, TTime_Data *ttime_ptr, bool ab_flag, int dir, double tod_a, Use_Type type)	The function used to check the traveler's eligibility and call the Update_Dtoll_Imp function to update the dynamic toll
/Router/Drive_Path.cpp	bool Router::Write_Dtoll()	Write the dynamic toll to the output file
/Router/Drive_Path.cpp	int Router::Drive_Path (Use_Type type, bool best_flag)	Modified impedance function to accept VOT and dynamic tolls
/Router/Read_Activity.cpp	double Router::Get_VOT (Population_Data *pp_ptr, Activity_File *activity_file, Household_Data *hh_ptr)	Calculate the value of time
/Router/Read_Activity.cpp	bool Router::Write_Vot (void)	Write the value of time to the value of time output file
Syslib/Include/Vot_File.hpp	class VOT_Data : public Class_Index	Define the VOT_Data class
Syslib/Include/Vot_File.hpp	class Votoutput_Array : public Class_Array	Define the Votoutput_Array class
Syslib/Include/Vot_File.hpp	class Vot_File : public Ext_File,	Define the Vot_File

	public Static_Scale	class
Syslib/Include/Vot_File.hpp	class VOTCoefficients_Data : public Class_Index	Define the VOTCoefficients_Data class
Syslib/Include/Vot_File.hpp	class VOTCoefficients_Array : public Class_Array	Define the VOTCoefficients_Array class
Syslib/Include/Vot_File.hpp	class VOTCoefficients_File : public Db_Header	Define the VOTCoefficients_File class
Syslib/Include/Dtoll_File.hpp	class DToll_Data : public Class_Index	Define the DToll_Data class
Syslib/Include/Dtoll_File.hpp	class DToll_Array : public Class_Array	Define the class DToll_Array class
Syslib/Include/Dtoll_File.hpp	class DToll_Output_Data : public Class_Index	Define the DToll_Output_Data class
Syslib/Include/Dtoll_File.hpp	class DToll_Output_Array : public Class_Array	Define the DToll_Output_Array class
Syslib/Include/Dtoll_File.hpp	class DToll_File : public Db_Header	Define the DToll_File class
Syslib/Include/Dtoll_File.hpp	class Lookuptable_Data : public Class_Index	Define the Lookuptable_Data class
Syslib/Include/Dtoll_File.hpp	class Lookuptable_Array : public Class_Array	Define the Lookuptable_Array class
Syslib/Include/Dtoll_File.hpp	class Lookuptable_File : public Db_Header	Define the Lookuptable_File class

Appendix B. Added/Modified Source Codes for VOT and Dynamic Tolls

< /Router/Drive_Path.cpp >

```
#include "Router.hpp"
#include "Toll_File.hpp"
#include "Dtoll_File.hpp"
#include "math.h"

#define NONTOLLLINK  -1000

//----------------------------------------------------------
//     Output_Dtoll
//----------------------------------------------------------
void Router::Output_Dtoll(int volume, int capacity,  double ratio, int basetoll, int
newtoll, DToll_Data* toll_ptr_cp)
{
    DToll_Output_Data *dtoll_output_ptr = NULL;

    int index = toll_ptr_cp->Link()+toll_ptr_cp->Start()+ toll_ptr_cp-
>End()+toll_ptr_cp->Node();

       if(!dtoll_output_data.Get_Index(index)){
       dtoll_output_ptr = (DToll_Output_Data *) dtoll_output_data.New_Record ();
    dtoll_output_ptr->Cap(capacity);
    dtoll_output_ptr->Vol(volume);
       dtoll_output_ptr->Link(toll_ptr_cp->Link());
       dtoll_output_ptr->Ratio(ratio);
    dtoll_output_ptr->End(Resolve(toll_ptr_cp->End()));
       dtoll_output_ptr->Start(Resolve(toll_ptr_cp->Start()));
    dtoll_output_ptr->Index(index);
    dtoll_output_ptr->Oldtoll(toll_ptr_cp->Toll());
       dtoll_output_ptr->Basetoll(basetoll);
    dtoll_output_ptr->Newtoll(newtoll);
       if (!dtoll_output_data.Add ()) {
              Print (2, "Adding data to dtoll output is unsucessful");

       }
       }
}

//----------------------------------------------------------
//     Base_toll
//----------------------------------------------------------
double Router::Base_Toll(int volume, int capacity, DToll_Data* toll_ptr_cp)
{
       Lookuptable_Data *lookuptable_ptr = NULL;
```

```
        double lowvc = 0, highvc = 0, toll = 0, computervc = 0,tmp =0;
        bool found = false;
    volume = 4*volume;
        tmp = (double)1/capacity;
        computervc = volume*tmp;
       //Print (2, "Hourly Volume =%d; Capacity =%d; Volume Capacity ratio is = %0.3f;",
volume,capacity, computervc);
        lookuptable_ptr = lookuptable_data.First();
        lowvc = lookuptable_ptr->Lowvc();
        highvc = lookuptable_ptr->Highvc();
        if(computervc >= lowvc && computervc < highvc){
                return lookuptable_ptr->Toll();
        }
        while((lookuptable_ptr=lookuptable_data.Next())!=NULL){
                lowvc = lookuptable_ptr->Lowvc();
                highvc = lookuptable_ptr->Highvc();
                if(computervc >= lowvc && computervc < highvc){
                        toll = lookuptable_ptr->Toll();
                        found = true;
                    break;
                }
        }
        if (found){
                return toll;
        }
    lookuptable_ptr=lookuptable_data.Last();
        lowvc = lookuptable_ptr->Lowvc();
        highvc = lookuptable_ptr->Highvc();
        if(computervc >= lowvc && computervc < highvc){
                return lookuptable_ptr->Toll();
        }
}

//---------------------------------------------------------------
//      Setup_dtoll_cp_array
//---------------------------------------------------------------
void Router::Setup_dtoll_cp_array (void)
{
        DToll_Data* toll_ptr = NULL, *toll_ptr_tmp = NULL;

        dtoll_array_cp = new DToll_Array;

        if (dtoll_array_cp ==NULL)
                      Error ("Not able to allocate memory for dtoll_array_copy records");
        toll_ptr_tmp = dtoll_data.First();
        if(toll_ptr_tmp->Key() == 1 || toll_ptr_tmp->Key() == -1){
                toll_ptr = dtoll_data_cp.New_Record (true);
                copy_dtoll_ptr (toll_ptr, toll_ptr_tmp);
                dtoll_data_cp.Add ();
        }

        //toll_ptr = dtoll_data_cp.New_Record (true);
        while((toll_ptr_tmp=dtoll_data.Next())!=NULL){
                if(toll_ptr_tmp->Key() == 1 || toll_ptr_tmp->Key() == -1){
                        toll_ptr = dtoll_data_cp.New_Record (true);
                        copy_dtoll_ptr (toll_ptr, toll_ptr_tmp);
                        dtoll_data_cp.Add ();
                }
        }

        toll_ptr_tmp = dtoll_data.Last();
        if(toll_ptr_tmp->Key() == 1|| toll_ptr_tmp->Key() == -1){
                toll_ptr = dtoll_data_cp.New_Record (true);
                copy_dtoll_ptr (toll_ptr, toll_ptr_tmp);
                dtoll_data_cp.Add ();
        }
}

//-----------------------------------------------------------
//      copy_dtoll_ptr
//-----------------------------------------------------------
void Router::copy_dtoll_ptr (DToll_Data* toll_ptr, DToll_Data* toll_ptr_tmp)
{
        toll_ptr->Index(toll_ptr_tmp->Index());
        toll_ptr->Link(toll_ptr_tmp->Link());
    toll_ptr->Start(toll_ptr_tmp->Start());
    toll_ptr->Key(toll_ptr_tmp->Key());
    toll_ptr->End(toll_ptr_tmp->End());
    toll_ptr->Use(toll_ptr_tmp->Use());
    toll_ptr->Toll(toll_ptr_tmp->Toll());
    toll_ptr->Node(toll_ptr_tmp->Node());
    toll_ptr->Notes(toll_ptr_tmp->Notes());
}

//-----------------------------------------------------------
//      Cal_Dtoll
//-----------------------------------------------------------
double Router::Cal_Dtoll(int volume, int capacity, DToll_Data* toll_ptr_cp)
{
        int key = 0; double toll = 0, basetoll = 0, oldtoll = 0, computervc = 0,tmp =0;
       key = toll_ptr_cp->Key();
       if(key==0 || key==-1)
              toll = toll_ptr_cp->Toll();
       else if (key==1){
     basetoll = Base_Toll(volume, capacity,toll_ptr_cp);
     oldtoll = toll_ptr_cp->Toll();
        if((basetoll-oldtoll)>=toll_increament_constant)
     toll = oldtoll + toll_increament_constant;
     else if ((basetoll-oldtoll)>-toll_decreament_constant)
     toll = oldtoll;
     else
     toll = oldtoll - toll_decreament_constant;

        volume = 4*volume;
        tmp = (double)1/capacity;
        computervc = volume*tmp;
        Output_Dtoll(volume, capacity, computervc, basetoll, toll, toll_ptr_cp);

       }
       else
       Print (2,"The DToll file key %d is not a pre-defined key (0,1,-1) ",key);

       return toll;

}

//--------------------------------------------------------
//     Update_Dtoll_Imp
//--------------------------------------------------------
double Router::Update_Dtoll_Imp(Link_Data *link_ptr,DToll_Data *toll_ptr, DToll_Data
*toll_ptr_cp, TTime_Data *ttime_ptr, bool ab_flag,int dir)
{
       int cap, vol,period_dtoll, linkconstant = 10, indexed_period = 0; double
dtoll=NONTOLLLINK; bool found = false;
          indexed_period = toll_ptr_cp->Link()* linkconstant +  dir + toll_ptr_cp-
>End();
          //int final_indexed_period = Time_Period (indexed_period - 1);
       //---- subtract 1 to adjust end time ---
       if (ab_flag){
                     cap = link_ptr->CAP_AB();
                     period_dtoll =Demand_Service::Time_Period (toll_ptr_cp->End()-1);
                     vol = ttime_ptr->Volume(period_dtoll);
                     //Print (2, "Hourly Volume =%d; Capacity =%d; Bnode = %d",
vol,cap,link_ptr->Anode());
                if(vol>0 && cap>0){
                    dtoll = Cal_Dtoll(vol,cap,toll_ptr_cp);
             toll_ptr->Toll(dtoll);
```

```
                    if(toll_ptr->Key() == 1)
                           toll_ptr->Key(0);
                    }
            }
          else{
                    cap = link_ptr->CAP_BA();
                    period_dtoll =Demand_Service::Time_Period (toll_ptr_cp->End()-1);
                    vol = ttime_ptr->Volume(period_dtoll);
                    //Print (2, "Hourly Volume =%d; Capacity =%d; Bnode = %d",
vol,cap,link_ptr->Anode());
                if(vol>0 && cap>0){
                    dtoll = Cal_Dtoll(vol,cap,toll_ptr_cp);
                    toll_ptr->Toll(dtoll);
                    //Print (2,"ba flag");
                    if(toll_ptr->Key() == 1)
                toll_ptr->Key(0);
                    }
          }

          return dtoll;
}

//----------------------------------------------------------
//     Cal_Write
//----------------------------------------------------------
double Router::Cal_Write(Link_Data *link_ptr, TTime_Data *ttime_ptr, bool ab_flag, int
dir, double tod_a, Use_Type type)
{
       double dtoll = NONTOLLLINK; bool found = false, permission = false;

          int NodeNumber = 0;

          DToll_Data *toll_ptr = NULL, *toll_ptr_cp = NULL;
          toll_ptr_cp = dtoll_data_cp.First();
          Node_Data *node_ptr;

          //return 10;

          int link=link_ptr->Link();
          permission = true;

          NodeNumber = toll_ptr_cp->Node();
          if (node_flag) {
                    node_ptr = node_data.Get (NodeNumber);
                    //if (node_ptr == NULL) Print(2, "Node_ptr error in drv path
Cal_write function");
                    NodeNumber = Renumber () ? node_data.Record_Index () : NodeNumber;
          }

        if(link_ptr->Link() == toll_ptr_cp->Link()&& toll_ptr_cp->Start () <= tod_a &&
tod_a < toll_ptr_cp->End () && permission && dir == NodeNumber){
                        toll_ptr = dtoll_data.Get(toll_ptr_cp->Index());
                        if(toll_ptr->Key() == 1)
                               dtoll = Update_Dtoll_Imp(link_ptr,toll_ptr, toll_ptr_cp,
ttime_ptr, ab_flag, dir);
                        else
                               dtoll = toll_ptr->Toll();
                        return dtoll;
        }

      while((toll_ptr_cp = dtoll_data_cp.Next())!=NULL){
               if(link_ptr->Link()==toll_ptr_cp->Link() && toll_ptr_cp->Start () <=
tod_a && tod_a < toll_ptr_cp->End () && permission && dir == NodeNumber){

                                    toll_ptr = dtoll_data.Get(toll_ptr_cp->Index());
                                    if(toll_ptr->Key() == 1)
                                           dtoll =
Update_Dtoll_Imp(link_ptr ,toll_ptr,toll_ptr_cp,ttime_ptr, ab_flag, dir);
                                    else
                                           dtoll = toll_ptr->Toll();
                                    found = true;
                                    break;
                }

               }
        if (found)
             return dtoll;
        toll_ptr_cp = dtoll_data_cp.Last();
        if(link_ptr->Link()==toll_ptr_cp->Link() && toll_ptr_cp->Start () <= tod_a &&
tod_a < toll_ptr_cp->End () && permission && dir == NodeNumber){
                       toll_ptr = dtoll_data.Get(toll_ptr_cp->Index());
                       if(toll_ptr->Key() == 1)
                            dtoll = Update_Dtoll_Imp(link_ptr, toll_ptr, toll_ptr_cp,
ttime_ptr, ab_flag, dir);
                       else
                            dtoll = toll_ptr->Toll();

        }

        return dtoll;
}

//-------------------------------------------------------
//      Write_Dtoll
```

```
//-------------------------------------------------------
bool Router::Write_Dtoll()
{
	Db_File *file = Network_Service::Network_Db_File (DTOLL);
	DToll_File *toll_file = (DToll_File *) file;
	Db_File toll_temp_file;   FILE *temp_file;

	DToll_Data *toll_ptr; //FILE* tem_toll_file;
	char snew_toll[256];  int ii=0, start, end;

	ii=toll_data.Num_Records();

    strcpy(snew_toll,file->Filename());
    toll_file->Close();

	toll_temp_file.File_Access (CREATE);
	toll_temp_file.Open(snew_toll);

	temp_file = toll_temp_file.File();
	fprintf(temp_file,"%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\n","INDEX",
"KEY","LINK","NODE","START","END","USE","TOLL","NOTES");


	for (int i=1;i<=dtoll_data.Num_Records();i++)
	{
		//ii=dtoll_data.Num_Records();
		toll_ptr=dtoll_data[i];
		start=Resolve(toll_ptr->Start());
	 end=Resolve(toll_ptr->End());
	    fprintf(temp_file,"%d\t%d\t%d\t%d\t%d\t%d\t%s\t%d\t%s\n", i, toll_ptr->Key(),
toll_ptr->Link(),toll_ptr->Node(),start,end,Use_Code(toll_ptr->Use()),toll_ptr-
>Toll(),toll_ptr->Notes());
	 //ii=toll_data.Record_Index();
	}
	fclose(temp_file);

	Activity_Data* activity_ptr = NULL;
    activity_ptr = activity_data.First();


	DToll_Output_Data *dtoll_output_ptr = NULL;
	dtoll_output_ptr = dtoll_output_data.First();
	if(dtoll_output_ptr != NULL){
		Print (2, "Link =%d; Hourly Volume =%d; Capacity =%d; V/C =
%0.3f",dtoll_output_ptr->Link(), dtoll_output_ptr->Vol() ,dtoll_output_ptr->Cap(),
dtoll_output_ptr->Ratio());
	      Print (2, "  Start =%d; End =%d; BaseToll = %d; OldToll = %d; NewToll =%d",
dtoll_output_ptr->Start(),dtoll_output_ptr->End(), dtoll_output_ptr->Basetoll(),
dtoll_output_ptr->Oldtoll(), dtoll_output_ptr->Newtoll());

       }
    while((dtoll_output_ptr = dtoll_output_data.Next())!=NULL){
       Print (2, "Link =%d; Hourly Volume =%d; Capacity =%d; V/C =
%0.3f;",dtoll_output_ptr->Link(), dtoll_output_ptr->Vol() ,dtoll_output_ptr->Cap(),
dtoll_output_ptr->Ratio());
           Print (2, "  Start =%d; End =%d; BaseToll = %d; OldToll = %d; NewToll =%d",
dtoll_output_ptr->Start(),dtoll_output_ptr->End(), dtoll_output_ptr->Basetoll(),
dtoll_output_ptr->Oldtoll(), dtoll_output_ptr->Newtoll());
    }
       /*if((dtoll_output_ptr=dtoll_output_data.Last())!=NULL){
        Print (2, " Link =%d; Hourly Volume =%d; Capacity =%d; V/C =
%0.3f;",dtoll_output_ptr->Link(), dtoll_output_ptr->Vol() ,dtoll_output_ptr->Cap(),
dtoll_output_ptr->Ratio());
           Print (2, "Start =%d; End =%d; BaseToll = %d; Old TOLL = %d;",
dtoll_output_ptr->Start(),dtoll_output_ptr->End(), dtoll_output_ptr->Basetoll(),
dtoll_output_ptr->Oldtoll());
       }*/
       return true;
}

//-----------------------------------------------------------
//     Drive_Path
//-----------------------------------------------------------

int Router::Drive_Path (Use_Type type, bool best_flag)
{
       int lnk, offset, offset_factor, period, approach, dir, use;
       unsigned cum_a, cum_b, max_cum, hi_cum;
       int tod_a, tod_b, time_a, time_b, penalty, pen_imp;
       int imp, imped, tim, time, variance, best_des, list;
       int x0, y0, x1, y1, x2, y2, dist, max_dist,alpha;
       int cost_a, cost_b, cost;  // for link capacity, link ID, volume
       double dx, dy;
       bool ab_flag = false, des_flag, hov_flag, hov_lanes, local;

       double dcost =0;

       Trip_End_Data *org_ptr, *des_ptr;
       Node_Data *node_ptr;
       Link_Data *link_ptr;
       TTime_Data *ttime_ptr;
       Link_Use_Data *use_ptr;
       TOD_Turn_Data *turn_ptr;
       Connect_Time *connect_ptr;
```

```
	Path_Data *path_ptr, path_root;
	Toll_Data *toll_ptr;

	// to enable SOV to be charged with TOLL, but HOV not
	if( activity_file->Passengers()>=1)
	 alpha=0;
	else
	 alpha=1;

	if (type == CAR) type = SOV;
	hov_flag = (type == HOV2 || type == HOV3 || type == HOV4);
	local = true;

	//---- initialize the org data ----
	org_ptr = org_array.First ();

	link_ptr = link_data [org_ptr->Link ()];

	if (!Use_Permission (link_ptr->Use (), type)) {
		use_flag = true;
		return (0);
	}
	cum_a = org_ptr->Cum ();
	tod_a = org_ptr->TOD ();
	cost_a = org_ptr->Cost ();
	time_a = org_ptr->Time ();

	cost_b = cost_a;
	time_b = time_a;

	if (tod_flag) {
		period = ttime_data.Period (tod_a);
	}

	best_des = 0;
	max_cum = MAX_INTEGER;

	//---- check for org and des on the same link ----
	des_flag = false;

	for (des_ptr = des_array.First (); des_ptr != NULL; des_ptr = des_array.Next ())
{
		if (org_ptr->Link () != des_ptr->Link ()) continue;

		//---- get direction and distance ----

		if (org_ptr->Offset () <= des_ptr->Offset ()) {
				dir = link_ptr->AB_Dir ();
				offset = des_ptr->Offset () - org_ptr->Offset ();
				ab_flag = true;
			} else {
				dir = link_ptr->BA_Dir ();
				offset = org_ptr->Offset () - des_ptr->Offset ();
				ab_flag = false;
			}
			if (dir == 0) {
				zero_flag = true;
				des_ptr->Link (0);
				continue;
			}
			offset_factor = ((offset << offset_roll) + (link_ptr->Length () >> 1)) / 
link_ptr->Length ();

			ttime_ptr = ttime_data [dir];

			//---- update delay ----

			if (tod_flag) {
				time = ttime_ptr->TTime (period);
			} else {
				time = ttime_ptr->Time0 ();
			}

		//---- calculate time, comment:originally calculating time is below 
calculating cost, since we need tod_b, as such it was moved above calculating cost ---

			tim = (time * offset_factor + offset_round) >> offset_roll;

			tod_b = tod_a + tim;

			if (tod_b < tod_a || tod_b > max_tod) {
				time_flag = true;
				continue;
			}

			//---- get cost data ----

			cost = 0;

			if (toll_flag) {
				for (list = ttime_ptr->Cost_List (); list; list = toll_ptr-
>TOD_List ()) {
					toll_ptr = toll_data [list];
					if (toll_ptr->Start () <= tod_a && tod_a < toll_ptr->End ())
{
                                        if (Use_Permission (toll_ptr->Use(), type)) {
                                                cost = toll_ptr->Toll ();

                                                break;
                                        }
                                }
                        }
                }

            if(dtoll_flag){
                        dcost=Cal_Write(link_ptr,ttime_ptr,ab_flag,link_ptr-
>Bnode(),tod_a,type);
                        if(dcost == NONTOLLLINK){
                                alpha = 0;
                        }
                        else
                  alpha = 1;
                }
                if (toll_flag)
                        imped = (link_ptr->Length () * value_distance +
(int)(cost*value_cost) +
                                time * value_time + value_round) >> value_roll;
                else if (dtoll_flag)
             imped = (link_ptr->Length () * value_distance + (int)(dcost/vot*alpha) +
                                time * value_time + value_round) >> value_roll;
                else
             imped = (link_ptr->Length () * value_distance + (int)(cost*value_cost) +
                                time * value_time + value_round) >> value_roll;

                if (imped < 0) continue;

                if (random_flag) {
                        variance = (imped * random_imped + 50) / 100;
                        if (variance > 0) {
                                imped = (int) (imped + variance * (random.Probability () -
0.5) + 0.5);
                        }
                }

                //---- check the cumulative impedance ----

                imp = (imped * offset_factor + offset_round) >> offset_roll;
                if (imp < 1) {
                        if (imp < 0) continue;
                        imp = 1;
                }
                cum_b = cum_a + imp;

                if (cum_b < 0 || cum_b >= max_cum || (cum_b >= des_ptr->Cum () &&
des_ptr->Cum ())) continue;

                des_ptr->Cum (cum_b);
                des_ptr->TOD (tod_b);
                des_ptr->Cost (cost_b);
                des_ptr->Time (time_b);
                des_ptr->Prev (dir);

                link_path [dir]->Prev (0);

                des_flag = true;

                if (best_flag) {
                        max_cum = cum_b;
                        best_des = des_array.Record_Index ();
                }
            }

            //---- update the max cum ----

            if (best_flag) {
                if (best_des > 0) return (best_des);
                if (zero_flag) return (0);
            } else if (des_flag || zero_flag) {
                hi_cum = 0;

                for (des_ptr = des_array.First (); des_ptr != NULL; des_ptr =
des_array.Next ()) {
                        if (des_ptr->Link () > 0 && des_ptr->Cum () > hi_cum) hi_cum =
des_ptr->Cum ();
                }
                if (hi_cum == 0) return (0);
                if (hi_cum < max_cum) max_cum = hi_cum;
            }

            //---- initialize the path building ----

            link_path.Zero_Fill ();

            first_ptr = last_ptr = &path_root;

            first_ptr->Next (0);

            //---- process each direction ----
```

```
	for (lnk=0; lnk < 2; lnk++) {

		//---- get direction data ----

		if (lnk == 0) {
			dir = link_ptr->AB_Dir ();
			offset = link_ptr->Length () - org_ptr->Offset ();
		} else {
			dir = link_ptr->BA_Dir ();
			offset = org_ptr->Offset ();
		}
		if (dir == 0) continue;

		offset_factor = ((offset << offset_roll) + (link_ptr->Length () >> 1)) /
link_ptr->Length ();

		ttime_ptr = ttime_data [dir];

		//---- check time of day restrictions ----

		hov_lanes = false;

		if (ttime_ptr->TOD_List () > 0) {
			use = link_ptr->Use ();

			for (list = ttime_ptr->TOD_List (); list; list = use_ptr->TOD_List
()) {
				use_ptr = link_use_data [list];

				if (use_ptr->Start () <= tod_a && tod_a < use_ptr->End ())
{
					use = use_ptr->Use ();
					hov_lanes = (hov_flag && use_ptr->HOV_Lanes () > 0
&&
							use_ptr->HOV_Lanes () <
use_ptr->Lanes ());
					break;
				}
			}
			if (!Use_Permission (use, type)) {
				use_flag = true;
				continue;
			}
		}

		//---- update delay ----
		if (tod_flag && !hov_lanes) {
			time = ttime_ptr->TTime (period);
		} else {
			time = ttime_ptr->Time0 ();
		}

		//---- calculate time ----

		tim = (time * offset_factor + offset_round) >> offset_roll;

		//---- check the time schedule ----

		tod_b = tod_a + tim;

		if (tod_b < tod_a || tod_b > max_tod) {
			time_flag = true;
			continue;
		}

		//---- get cost data ----

		cost = 0;

		if (toll_flag) {
			for (list = ttime_ptr->Cost_List (); list; list = toll_ptr-
>TOD_List ()) {
				toll_ptr = toll_data [list];

				if (toll_ptr->Start () <= tod_a && tod_a < toll_ptr->End ())
{
					if (Use_Permission (toll_ptr->Use (), type)) {
						cost = toll_ptr->Toll ();

						break;
					}
				}
			}
		}

	    if(dtoll_flag){
			dcost=Cal_Write(link_ptr,ttime_ptr,ab_flag,link_ptr-
>Bnode(),tod_a,type);
			if(dcost == NONTOLLLINK){
				alpha = 0;
			}
			else
```

```
                alpha = 1;
              }
              if (toll_flag)
                     imped = (link_ptr->Length () * value_distance +
(int)(cost*value_cost) +
                            time * value_time + value_round) >> value_roll;
              else if (dtoll_flag)
            imped = (link_ptr->Length () * value_distance + (int)(dcost/vot*alpha) +
                            time * value_time + value_round) >> value_roll;
        else
            imped = (link_ptr->Length () * value_distance + (int)(cost*value_cost) +
                            time * value_time + value_round) >> value_roll;

              if (imped < 0)        continue;

              imp = (imped * offset_factor + offset_round) >> offset_roll;
              if (imp < 1) {
                     if (imp < 0) continue;
                     imp = 1;
              }

              cum_b = cum_a + imp;

              path_ptr = link_path [dir];

              if (cum_b < 0 || cum_b >= max_cum || (cum_b >= path_ptr->Cum () &&
path_ptr->Cum ())) continue;

              if (path_ptr->Next () == 0 && last_ptr != path_ptr) {
                     last_ptr->Next (dir);
                     last_ptr = path_ptr;
              }
              path_ptr->Cum (cum_b);
              path_ptr->TOD (tod_b);
              path_ptr->Cost (cost_b);
              path_ptr->Time (time_b);
              path_ptr->Prev (-1);
       }
       if (first_ptr->Next () == 0) return (0);

       //---- calculate the org-des distance ----

       if (distance_flag || local_flag) {
              offset = org_ptr->Offset ();
              offset_factor = ((offset << offset_roll) + (link_ptr->Length () >> 1)) /
link_ptr->Length ();
```

```
		node_ptr = node_data [link_ptr->Anode ()];

		x0 = node_ptr->X ();
		y0 = node_ptr->Y ();

		node_ptr = node_data [link_ptr->Bnode ()];

		x0 += ((node_ptr->X () - x0) * offset_factor + offset_round) >>
offset_roll;
		y0 += ((node_ptr->Y () - y0) * offset_factor + offset_round) >>
offset_roll;

		//---- find the furthest destination ----

		max_dist = 0;
		x1 = y0;
		y1 = y0;

		for (des_ptr = des_array.First (); des_ptr != NULL; des_ptr =
des_array.Next ()) {
			if (des_ptr->Link () == 0) continue;

			link_ptr = link_data [des_ptr->Link ()];

			offset = des_ptr->Offset ();
			offset_factor = ((offset << offset_roll) + (link_ptr->Length () >>
1)) / link_ptr->Length ();

			node_ptr = node_data [link_ptr->Anode ()];

			x2 = node_ptr->X ();
			y2 = node_ptr->Y ();

			node_ptr = node_data [link_ptr->Bnode ()];

			x2 += ((node_ptr->X () - x2) * offset_factor + offset_round) >>
offset_roll;
			y2 += ((node_ptr->Y () - y2) * offset_factor + offset_round) >>
offset_roll;

			dx = x0 - x2;
			dy = y0 - y2;

			dist = (int) (sqrt (dx * dx + dy * dy) + 0.5);

			if (dist > max_dist) {
				x1 = x2;
```

```
                y1 = y2;
                max_dist = dist;
            }
        }
        dist = max_dist;
        max_dist = (dist * max_ratio + 50) / 100;

        if (max_dist < min_distance) {
            max_dist = min_distance;
        } else if (max_dist > max_distance) {
            max_dist = max_distance;
        }
        max_dist += dist;
    }

    //---- build the path ----

    for (;;) {
        approach = first_ptr->Next ();
        if (approach <= 0) break;

        first_ptr->Next (-1);

        first_ptr = link_path [approach];

        //---- check the cumulative impedance ----

        cum_a = first_ptr->Cum ();

        if (cum_a >= max_cum) continue;

        //---- check the activity schedule ----

        tod_a = first_ptr->TOD ();

        if (tod_a < 0 || tod_a >= max_tod) {
            time_flag = true;
            continue;
        }
        time_a = first_ptr->Time ();
        cost_a = first_ptr->Cost ();

        time_b = time_a;

        if (tod_flag) {
            period = ttime_data.Period (tod_a);
        }
```

```
//---- check the circuity ----

if (distance_flag || local_flag) {
	ttime_ptr = ttime_data [approach];

	link_ptr = link_data [ttime_ptr->Link ()];

	if (ttime_ptr->Dir ()) {
		node_ptr = node_data [link_ptr->Anode ()];
	} else {
		node_ptr = node_data [link_ptr->Bnode ()];
	}

	x2 = node_ptr->X ();
	y2 = node_ptr->Y ();

	dx = x0 - x2;
	dy = y0 - y2;

	dist = (int) (sqrt (dx * dx + dy * dy) + 0.5);

	local = (local_flag && dist <= local_distance);

	dx = x1 - x2;
	dy = y1 - y2;

	dir = (int) (sqrt (dx * dx + dy * dy) + 0.5);

	if (local_flag && dir <= local_distance) local = true;

	dist += dir;

	if (distance_flag && dist > max_dist) {
		dist_flag = true;
		continue;
	}
}

//---- process each link leaving the approach link ----

for (dir = drive_list [approach]; dir; dir = connect_ptr->Next_Rec ()) {

	penalty = pen_imp = 0;
	connect_ptr = connect_time [dir];

	//---- check for time of day prohibitions ----
```

```
                    for (list = connect_ptr->TOD_List (); list; list = turn_ptr-
>TOD_List ()) {
                            turn_ptr = tod_turn [list];

                           if (turn_ptr->Start () <= tod_a && tod_a < turn_ptr->End ())
{
                                   if (turn_ptr->Use () == 0 || Use_Permission
(turn_ptr->Use (), type)) break;
                            }
                    }
                    if (list) {
                            if (turn_ptr->Penalty () == 0) continue;
                            penalty = turn_ptr->Penalty ();
                    }

                    //--- get the link ---

                    dir = connect_ptr->Dir_Index ();

                    ttime_ptr = ttime_data [dir];

                    lnk = ttime_ptr->Link ();

                    ab_flag = (ttime_ptr->Dir () == 0);

                    link_ptr = link_data [lnk];

                    if(link_ptr->Link() ==5)
                            int directiontemp = ttime_ptr->Dir ();
                    //---- check the local access restrictions ----

                    if (!local && link_ptr->Type () >= local_type && link_ptr->Type ()
<= LOCAL) {

                            //**** problem flag ****
                            continue;
                    }

                    //---- check for time of day restrictions ----

                    use = link_ptr->Use ();
                    hov_lanes = false;

                    for (list = ttime_ptr->TOD_List (); list; list = use_ptr->TOD_List
()) {

                            use_ptr = link_use_data [list];
```

```
                                if (use_ptr->Start () <= tod_a && tod_a < use_ptr->End ())
{
                                        use = use_ptr->Use ();
                                        hov_lanes = (hov_flag && use_ptr->HOV_Lanes () > 0
&&
                                                                use_ptr->HOV_Lanes () <
use_ptr->Lanes ());
                                        break;
                                }
                        }

                        //---- check the vehicle type ----

                        if (!Use_Permission (use, type)) {
                                use_flag = true;
                                continue;
                        }

                        //---- get direction data ----

                        if (tod_flag && !hov_lanes) {
                                time = ttime_ptr->TTime (period);
                        } else {
                                time = ttime_ptr->Time0 ();
                        }

                        //---- update the connection penalty -----

                        if (tod_flag) {
                                penalty += connect_ptr->TTime (period);
                        }
                        if (penalty > 0) {
                                pen_imp = (penalty * value_time + value_round) >>
value_roll;
                        } else if (penalty < 0) {
                                pen_imp = -((-penalty * value_time + value_round) >>
value_roll);
                        }

                        //---- add turn impedance penalty ----

                        if (turn_flag) {
                                if (connect_ptr->Type () == LEFT) {
                                        imp = left_imped;
                                } else if (connect_ptr->Type () == RIGHT) {
                                        imp = right_imped;
                                } else if (connect_ptr->Type () == UTURN) {
```

```
                                imp = uturn_imped;
                        } else {
                                imp = 0;
                        }
                        if (pen_imp <= 0) {
                                pen_imp += imp;
                        } else if (pen_imp < imp) {
                                pen_imp = imp;
                        }
                }

                //---- get cost data ----

                cost = 0;

                if (toll_flag) {
                        for (list = ttime_ptr->Cost_List (); list; list = toll_ptr-
>TOD_List ()) {
                                toll_ptr = toll_data [list];

                                if (toll_ptr->Start () <= tod_a && tod_a <
toll_ptr->End ()) {
                                        if (Use_Permission (toll_ptr->Use (), type))
{
                                                cost = toll_ptr->Toll ();

                                                break;
                                        }
                                }
                        }
                }

            if(dtoll_flag){

                   // if(link_ptr->Link() ==12103)
                   //  Print (2, "tod_a =%d",tod_a);

                        dcost=Cal_Write(link_ptr,ttime_ptr,ab_flag,link_ptr-
>Bnode(),tod_a,type);
                    if(dcost == NONTOLLLINK){
                                alpha = 0;
                        }
                        else
                                alpha = 1;
             }
           if (toll_flag)
                imped = (link_ptr->Length () * value_distance +
(int)(cost*value_cost) +
                              time * value_time + value_round) >> value_roll;
                 else if (dtoll_flag){

                // int vol = ttime_ptr->Volume(period);

              //Print (2, "vot= %.2f ; dcot = %.2f ; volume = %d; capacity_AB = %d;
period = %d, direction= %d",vot, dcost, vol, link_ptr->CAP_AB(), period, dir);

          imped = (link_ptr->Length () * value_distance + (int)(dcost/vot*alpha) +
                              time * value_time + value_round) >> value_roll;



                 //if(link_ptr->Link() ==8)
                 //Print (2, "link 8 cost= %0.2f; period= %d, dir =%d; vol =%d",dcost,
period, dir, vol);

                 }
          else
            imped = (link_ptr->Length () * value_distance + (int)(cost*value_cost) +
                              time * value_time + value_round) >> value_roll;

                       if (imped < 0) continue;

                       imp = imped + pen_imp;
                       if (imp < 1) {
                              if (imp < 0 && pen_imp >= 0) continue;
                              imp = 1;
                       }
                       if (random_flag) {
                              variance = (imp * random_imped + 50) / 100;
                              if (variance > 0) {
                                     variance = (int) (imp + variance *
(random.Probability () - 0.5) + 0.5);

                                     if (pen_imp == 0) {
                                            imped = variance;
                                     } else {
                                            imped = (imped * variance + (imp >> 1)) /
imp;
                                            pen_imp = variance - imped;
                                     }
                              }
                       }
                       path_ptr = link_path [dir];
```

```
				cost_b = cost_a + cost;

				//---- check for the trip end ----

				des_flag = false;

				for (des_ptr = des_array.First (); des_ptr != NULL; des_ptr =
des_array.Next ()) {
					if (lnk != des_ptr->Link ()) continue;

					offset = (ab_flag) ? des_ptr->Offset () : link_ptr->Length
() - des_ptr->Offset ();

					offset_factor = ((offset << offset_roll) + (link_ptr-
>Length () >> 1)) / link_ptr->Length ();

					//---- calculate the time ----

					tim = (time * offset_factor + offset_round) >> offset_roll;
					if (tim < 0) {
						time_flag = true;
						continue;
					}
					tim += penalty;
					if (tim < 0 && penalty < 0) tim = 1;

					//---- check the time schedule ----

					tod_b = tod_a + tim;

					if (tod_b < tod_a || tod_b > max_tod) {
						time_flag = true;
						continue;
					}

					//---- check the cumulative impedance ----

					imp = ((imped * offset_factor + offset_round) >>
offset_roll) + pen_imp;
					if (imp < 1) {
						if (imp < 0 && pen_imp >= 0) continue;
						imp = 1;
					}
					cum_b = cum_a + imp;

					if (cum_b < 0 || cum_b >= max_cum || (cum_b >= des_ptr->Cum
() && des_ptr->Cum ())) continue;
```

```
			des_ptr->Cum (cum_b);
			des_ptr->TOD (tod_b);
			des_ptr->Cost (cost_b);
			des_ptr->Time (time_b);
			des_ptr->Prev (dir);

			path_ptr->Prev (approach);

			des_flag = true;

			if (best_flag) {
				max_cum = cum_b;
				best_des = des_array.Record_Index ();
			}
		}

		//---- update the max cum ----

		if (des_flag && !best_flag) {
			hi_cum = 0;

			for (des_ptr = des_array.First (); des_ptr != NULL; des_ptr
= des_array.Next ()) {
				if (des_ptr->Cum () > hi_cum) hi_cum = des_ptr->Cum
();
			}
			if (hi_cum > 0 && hi_cum < max_cum) max_cum = hi_cum;
		}

		//---- check the restricted access ----

		if (Use_Permission (use, RESTRICTED)) {
			use_flag = true;
			continue;
		}

		//---- check the time ----

		time += penalty;
		if (time < 0 && penalty < 0) {
			time = 1;
		}
		tod_b = tod_a + time;

		if (tod_b < 0 || tod_b >= max_tod) {
			time_flag = true;
```

```
					continue;
				}

				//---- check the cumulative impedance at B ----

				imp = imped + pen_imp;
				if (imp < 1) {
					if (imp < 0 && pen_imp >= 0) continue;
					imp = 1;
				}
				cum_b = cum_a + imp;

				if (cum_b < 0 || cum_b >= max_cum || (cum_b >= path_ptr->Cum () && 
path_ptr->Cum ())) continue;

				if (path_ptr->Next () == -1) {
					path_ptr->Next (first_ptr->Next ());
					first_ptr->Next (dir);
				} else if (path_ptr->Next () == 0 && last_ptr != path_ptr) {
					last_ptr->Next (dir);
					last_ptr = path_ptr;
				}
				path_ptr->Cum (cum_b);
				path_ptr->TOD (tod_b);
				path_ptr->Cost (cost_b);
				path_ptr->Time (time_b);
				path_ptr->Prev (approach);
			}
		}

		return (best_des);
}
```

< /Router/Read_Activity.cpp >

```
//***********************************************************
//     Read_Activity.cpp - Read the Activity File
//***********************************************************

#include "Router.hpp"

//-----------------------------------------------------------
//     calcualte the value of time
//-----------------------------------------------------------

double Router::Get_VOT (Population_Data *pp_ptr, Activity_File *activity_file,
Household_Data *hh_ptr )
{
   int age=0; int activitypurpose=0; int i=0; int hourtosecondconstant=3600,
dollartocentsconstant=100;

   double BetaT=0.8; double Betaage=-0.7; double employee=0.1;  double BetaC=0.06,
Betawork=0.02;
   double income=0.2; double value_for_income_group=0;
   double value_of_time=0, activitypurposecoefficient=1;

   double income_work_activity[]={1.00, 4.33,7.66,11.00,19.00,27.00, 35.00};
   double income_non_work_activity[]={1.00,1.17,1.33,1.50,1.67,1.83,2.00};
   int portland_income_group[]={1,2,3,4,5,6,7};

   FILE *file;

   VOTCoefficients_Data *votcoefficients_ptr = NULL;

   VOT_Data* votoutput_ptr = NULL;

   if(pp_ptr->Age ()>=45)
   age=1;
   else
   age=0;

   if(activity_file->Purpose()==1||activity_file->Purpose()==9|| activity_file-
>Purpose()==11 || activity_file->Purpose()==17 ||activity_file->Purpose()==18 ||
activity_file->Purpose()==19)
   activitypurpose=1;
   else
   activitypurpose=0;

   i=hh_ptr->Income();
   if(activitypurpose==1)
```

```
              value_for_income_group=income_work_activity[i-1];
    else
              value_for_income_group=income_non_work_activity[i-1];

        votcoefficients_ptr = (VOTCoefficients_Data*)votcoefficients_data.Get(1);

        if(votcoefficients_ptr!=NULL){
              BetaT = votcoefficients_ptr->Neumeconstant();
         Betaage = votcoefficients_ptr->Agecoefficient();
         employee = votcoefficients_ptr->Employeecoefficient();
         income = votcoefficients_ptr->Incomecoefficient();
         BetaC = votcoefficients_ptr->Denomconstant();
         Betawork = votcoefficients_ptr->Denworkcoefficient();
              activitypurposecoefficient = votcoefficients_ptr->Neuworkcoefficient();
    }

        value_of_time=(BetaT+Betaage*age+ employee*pp_ptr-
>Work()+activitypurposecoefficient*activitypurpose+income*value_for_income_group)/(Beta
C+Betawork*activitypurpose);//+BetaTpaid*file->income ()
    pp_ptr->VOT(value_of_time/hourtosecondconstant);

        if(vot_file_create_flag){
              //file = vot_file.File();
              //fprintf (file,
"%0.2f₩t%d₩t%d₩t%d₩t%d₩t%d₩t%d₩t%d₩n",value_of_time,activity_file-
>Household(),activity_file->Person(),activity_file->Activity(),pp_ptr->Age (),hh_ptr-
>Income(),pp_ptr->Work(),activitypurpose);
        votoutput_ptr = vot_output_data.New_Record (true);
        if (votoutput_ptr==NULL)
                     Error ("Not able to allocate memory for vot_output_data records");

        votoutput_ptr->Vot(value_of_time);
    votoutput_ptr->Hhold_ID(activity_file->Household());
    votoutput_ptr->Person_ID(activity_file->Person());
    votoutput_ptr->Activity_ID(activity_file->Activity());
    votoutput_ptr->Age(pp_ptr->Age ());
        votoutput_ptr->Income_group(hh_ptr->Income());
    votoutput_ptr->Employed(pp_ptr->Work());
    votoutput_ptr->Activity_purpose(activitypurpose);
        vot_output_data.Add ();
        }
        value_of_time=value_of_time/hourtosecondconstant*dollartocentsconstant;
        return value_of_time;
}

//---------------------------------------------------------
//      Write_Vot
```

```
//-------------------------------------------------------

bool Router::Write_Vot (void)
{

	FILE *file;

	VOT_Data *vot_ptr = NULL;

	file = vot_file.File();

    for (int i=1;i<=vot_output_data.Num_Records();i++)
	{
		vot_ptr=vot_output_data[i];
		fprintf (file, "%0.2f₩t%d₩t%d₩t%d₩t%d₩t%d₩t%d₩t%d₩n",vot_ptr-
>Vot(),vot_ptr->Hhold_ID(),vot_ptr->Person_ID(),vot_ptr->Activity_ID(),vot_ptr-
>Age(),vot_ptr->Income_group(),vot_ptr->Employed(),vot_ptr->Activity_purpose());

	}
	fclose(file);

	return true;

}
```

< Syslib/Include/Vot_File.hpp >

```
//**********************************************************
//     Vot_File.hpp - Vot File Input/Output
//**********************************************************

#ifndef VOT_FILE_HPP
#define VOT_FILE_HPP

#include "Static_Service.hpp"
#include "Ext_File.hpp"
#include "stdlib.h"
#include "Db_Header.hpp"
#include "Class_Index.hpp"
#include "Class_Array.hpp"

//---------------------------------------------------------
//     VOT_Data class definition
//---------------------------------------------------------
class VOT_Data : public Class_Index
{
public:
       VOT_Data (int index = 0);
       virtual ~VOT_Data (void)     {}

       int   Index (void)                { return (ID ()); }
       double  Vot (void)                { return (vot); }
       int   Age (void)                  { return (age); }
       int   Income_group (void)         { return (income); }
       int   Activity_purpose (void)     { return (activitypurpose); }
       int   Employed (void)             { return (employed); }
       int   Hhold_ID (void)            { return (hholdID); }
       int   Person_ID (void)            { return (personID); }
       int   Activity_ID (void)          { return (activityID); }


       void Index (int value)            { ID (value); }
       void Vot (double value)           { vot = value; }
       void Age (int value)              { age = value; }
       void Income_group (int value)     { income = value; }
       void Activity_purpose (int value) { activitypurpose = value; }
       void Employed (int value)         { employed = value; }
       void Hhold_ID (int value)         { hholdID = value; }
       void Person_ID (int value)        { personID = value; }
    void Activity_ID(int value)     { activityID = value; }

private:
	double		vot;
	int		age;
	int		income;
    int		activitypurpose;
	unsigned short	employed;
	int		hholdID;
    int		personID;
	int		activityID;
};

//------------------------------------------------------------
//	Votoutput_Array class definition
//------------------------------------------------------------
class Votoutput_Array : public Class_Array
{
public:
	Votoutput_Array (int max_records = 0);

	bool Add (VOT_Data *data = NULL)  { return (Class_Array::Add (data)); }

	VOT_Data * New_Record (bool clear = false, int number = 1)
					   { return ((VOT_Data *)
Class_Array::New_Record (clear, number)); }

	VOT_Data * Record (int index)     { return ((VOT_Data *) Class_Array::Record
(index)); }
	VOT_Data * Record (void)          { return ((VOT_Data *) Class_Array::Record
()); }

	VOT_Data * Get (int index)         { return ((VOT_Data *) Class_Array::Get
(index)); }
	VOT_Data * Get_GE (int index)      { return ((VOT_Data *) Class_Array::Get_GE
(index)); }
	VOT_Data * Get_LE (int index)      { return ((VOT_Data *) Class_Array::Get_LE
(index)); }

	VOT_Data * First (void)           { return ((VOT_Data *) Class_Array::First
()); }
       VOT_Data * Next (void)            { return ((VOT_Data *) Class_Array::Next ()); }

       VOT_Data * Last (void)            { return ((VOT_Data *) Class_Array::Last ()); }
	VOT_Data * Previous (void)        { return ((VOT_Data *) Class_Array::Previous
()); }

	VOT_Data * First_Key (void)       { return ((VOT_Data *) Class_Array::First_Key
()); }
	VOT_Data * Next_Key (void)        { return ((VOT_Data *) Class_Array::Next_Key
()); }

	VOT_Data * operator[] (int index) { return (Record (index)); }
};

//---------------------------------------------------------
//	Vot_File Class definition
//---------------------------------------------------------

class  Vot_File : public Ext_File, public Static_Scale
{
public:
	Vot_File (Access_Type access = READ, Memory_Type memory = PRIVATE_MEMORY,
Sort_Type sort = TRAVELER_SORT);
	Vot_File (char *filename, Access_Type access = READ, Memory_Type memory =
PRIVATE_MEMORY, Sort_Type sort = TRAVELER_SORT);
	~Vot_File ();

private:
	bool Setup_Record (void);
	void Setup (Memory_Type memory, Sort_Type sort);

	bool allocate_memory, time_sort;
	int num_record, num_plan, num_traveler, num_trip, max_data;

	VOT_Data *vot;
};

//---------------------------------------------------------
//	VOTCoefficients_Data class definition
//---------------------------------------------------------
class VOTCoefficients_Data : public Class_Index
{
public:
	VOTCoefficients_Data (int index = 0);
	virtual ~VOTCoefficients_Data (void)	{}

	int	Index(void)				{ return (ID()); }
	double Neumeconstant (void)				{ return
(neumeconstant); }
	double Denomconstant (void)				{ return
(denomconstant); }
	double Agecoefficient (void)			{ return (agecoefficient); }
	double Neuworkcoefficient (void)		{ return
(neuworkcoefficient); }
	double Denworkcoefficient (void)		{ return
(denworkcoefficient); }
```

```
    double Employeecoefficient (void)                    { return
(employeecoefficient); }
    double Incomecoefficient (void)                   { return (incomecoefficient); }

        void Index(int value)                         { ID (value); }
        void Neumeconstant (double value)                   { neumeconstant = value; }
        void Denomconstant (double value)                   { denomconstant = value; }
        void Agecoefficient (double value)                  { agecoefficient = value; }
        void Neuworkcoefficient (double value)             { neuworkcoefficient = value; }
        void Denworkcoefficient (double value)             { denworkcoefficient = value; }
        void Employeecoefficient (double value)    { employeecoefficient = value; }
        void Incomecoefficient (double value)               { incomecoefficient = value; }

private:
        double              neumeconstant;
        double              denomconstant;
        double              agecoefficient;
        double              neuworkcoefficient;
        double              denworkcoefficient;
        double              employeecoefficient;
        double              incomecoefficient;
};

//---------------------------------------------------------
//      VOTCoefficients_Array class definition
//---------------------------------------------------------
class VOTCoefficients_Array : public Class_Array
{
public:
        VOTCoefficients_Array (int max_records = 0);

        bool Add (VOTCoefficients_Data *data = NULL)  { return (Class_Array::Add
(data)); }

        VOTCoefficients_Data * New_Record (bool clear = false, int number = 1)
                                        { return ((VOTCoefficients_Data *)
Class_Array::New_Record (clear, number)); }

        VOTCoefficients_Data * Record (int index)     { return ((VOTCoefficients_Data *)
Class_Array::Record (index)); }
        VOTCoefficients_Data * Record (void)          { return ((VOTCoefficients_Data *)
Class_Array::Record ()); }

        VOTCoefficients_Data * Get (int id)         { return ((VOTCoefficients_Data *)
Class_Array::Get (id)); }
        VOTCoefficients_Data * Get_GE (int id)      { return ((VOTCoefficients_Data *)
Class_Array::Get_GE (id)); }
```

```
	VOTCoefficients_Data * Get_LE (int id)      { return ((VOTCoefficients_Data *)
Class_Array::Get_LE (id)); }

	VOTCoefficients_Data * First (void)          { return ((VOTCoefficients_Data *)
Class_Array::First ()); }
	VOTCoefficients_Data * Next (void)           { return ((VOTCoefficients_Data *)
Class_Array::Next ()); }

	VOTCoefficients_Data * Last (void)           { return ((VOTCoefficients_Data *)
Class_Array::Last ()); }
	VOTCoefficients_Data * Previous (void)       { return ((VOTCoefficients_Data *)
Class_Array::Previous ()); }

	VOTCoefficients_Data * First_Key (void)      { return ((VOTCoefficients_Data *)
Class_Array::First_Key ()); }
	VOTCoefficients_Data * Next_Key (void)       { return ((VOTCoefficients_Data *)
Class_Array::Next_Key ()); }

	VOTCoefficients_Data * operator[] (int index) { return (Record (index)); }
};

//---------------------------------------------------------
//	VOTCoefficients_File Class definition
//---------------------------------------------------------
class  VOTCoefficients_File : public Db_Header
{
public:

	VOTCoefficients_File (Access_Type access = READ, Format_Type format =
DEFAULT_FORMAT, bool notes_flag = false);
	VOTCoefficients_File (char *filename, Access_Type access = READ, Format_Type
format = DEFAULT_FORMAT, bool notes_flag = false);
	virtual ~VOTCoefficients_File (void);

	int	Id(void)                        { Get_Field (id, &value); return
(value); }
	double Neumeconstant (void)                        { Get_Field
(neumeconstant, &lvalue); return (lvalue); }
	double Denomconstant (void)                        { Get_Field
(denomconstant, &lvalue); return (lvalue); }
	double Agecoefficient (void)               { Get_Field (agecoefficient,
&lvalue); return (lvalue); }
	double Neuworkcoefficient (void)           { Get_Field
(neuworkcoefficient, &lvalue); return (lvalue); }
	double Denworkcoefficient (void)           { Get_Field
(denworkcoefficient, &lvalue); return (lvalue); }
    double Employeecoefficient (void)          { Get_Field
(employeecoefficient, &lvalue); return (lvalue);} // to read new marker toll file
    double Incomecoefficient (void)                    { Get_Field (incomecoefficient,
&lvalue); return (lvalue);}  // to read new node toll file

       void Id(int value)                          { Put_Field (id, value); }
       void Neumeconstant (double value)                 { Put_Field (neumeconstant,
value); }
       void Denomconstant (double value)                 { Put_Field (denomconstant,
value); }
       void Agecoefficient (double value)                { Put_Field (agecoefficient,
value); }
       void Neuworkcoefficient (double value)            { Put_Field
(neuworkcoefficient, value); }
       void Denworkcoefficient (double value)            { Put_Field
(denworkcoefficient, value); }
       void Employeecoefficient (double value)    { Put_Field (employeecoefficient,
value); }
       void Incomecoefficient (double value)            { Put_Field (incomecoefficient,
value); }

       virtual bool Create_Fields (void);

protected:
       virtual bool Set_Field_Numbers (void);

private:
       void Setup (void);

       double lvalue;

       int neumeconstant , denomconstant , agecoefficient , neuworkcoefficient ,
denworkcoefficient,  employeecoefficient , incomecoefficient, id, value ; // for
putting marker option in the toll file
};

#endif
```

< Syslib/Include/Dtoll_File.hpp >

```
//**********************************************************
//      Toll_File.hpp - Toll File Input/Output
//**********************************************************
#ifndef DTOLL_FILE_HPP
#define DTOLL_FILE_HPP

#include "Static_Service.hpp"
#include "Ext_File.hpp"
#include "stdlib.h"
#include "Db_Header.hpp"
#include "Class_Index.hpp"
#include "Class_Array.hpp"

//---------------------------------------------------------
//      DToll_Data class definition
//---------------------------------------------------------
class DToll_Data : public Class_Index
{
public:
       DToll_Data (int index = 0);
       virtual ~DToll_Data (void)      {}

       int  Index (void)                { return (ID ()); }
       int  Link (void)                 { return (link); }
       int  Start (void)                { return (start); }
       int  Key (void)                  { return (key); }
       int  End (void)                  { return (end); }
       int  Use (void)                  { return (use); }
       int  Toll (void)                 { return (toll); }
       int  Node (void)                 { return (node); }  //-- for reading the
marker value  0- no need to update, 1- need to update
       char* Notes (void)               { return (notes); }  //-- for reading the
marker value  0- no need to update, 1- need to update

       void Index (int value)           { ID (value); }
       void Link (int value)            { link = value; }
       void Start (int value)           { start = value; }
       void Key (int value)             { key = value; }
       void End (int value)             { end = value; }
       void Use (int value)             { use = (unsigned short)value; }
       void Toll (int value)            { toll = value; }
       void Node (int value)            { node = value; } // for reading the marker
value  0- no need to update, 1- need to update
    void Notes(char* value)          { Copy_Data_notes (value); } // for reading the
marker value  0- no need to update, 1- need to update
```

```
protected:

    char* Copy_Data_notes (char * value);

private:
        int             start;
        int             end;
        int             key;
    int             link;
        unsigned short use;
        int             toll;                   //---- cents ----
    int             node;             //--  for reading the node value
        char*           notes;            //-- for reading the notes value
};

//----------------------------------------------------------
//     DToll_Array class definition
//----------------------------------------------------------
class DToll_Array : public Class_Array
{
public:
        DToll_Array (int max_records = 0);

        bool Add (DToll_Data *data = NULL)  { return (Class_Array::Add (data)); }

        DToll_Data * New_Record (bool clear = false, int number = 1)
                                           { return ((DToll_Data *)
Class_Array::New_Record (clear, number)); }

        DToll_Data * Record (int index)     { return ((DToll_Data *) Class_Array::Record
(index)); }
        DToll_Data * Record (void)          { return ((DToll_Data *) Class_Array::Record
()); }

        DToll_Data * Get (int index)         { return ((DToll_Data *) Class_Array::Get
(index)); }
        DToll_Data * Get_GE (int index)      { return ((DToll_Data *)
Class_Array::Get_GE (index)); }
        DToll_Data * Get_LE (int index)      { return ((DToll_Data *)
Class_Array::Get_LE (index)); }

        DToll_Data * First (void)           { return ((DToll_Data *) Class_Array::First
()); }
        DToll_Data * Next (void)            { return ((DToll_Data *) Class_Array::Next
()); }
```

```
	DToll_Data * Last (void)		{ return ((DToll_Data *) Class_Array::Last
()); }
	DToll_Data * Previous (void)		{ return ((DToll_Data *)
Class_Array::Previous ()); }

	DToll_Data * First_Key (void)		{ return ((DToll_Data *)
Class_Array::First_Key ()); }
	DToll_Data * Next_Key (void)		{ return ((DToll_Data *)
Class_Array::Next_Key ()); }

	DToll_Data * operator[] (int index) { return (Record (index)); }
};

class DToll_Output_Data : public Class_Index
{
public:
	DToll_Output_Data (int index = 0);
	virtual ~DToll_Output_Data (void)	{}

	int  Index (void)			{ return (ID ()); }
	int  Link (void)			{ return (link); }
	int  Start (void)			{ return (start); }
	int  End (void)			{ return (end); }
	int  Vol (void)			{ return (vol); }
	int  Cap (void)			{ return (cap); }
	double Ratio (void)			{ return (ratio); }
	int  Oldtoll (void)			{ return (oldtoll); }
	int  Basetoll (void)			{ return (basetoll);}
	int  Newtoll (void)			{ return (newtoll); }

	void Index (int value)		{ ID (value); }
	void Link (int value)		{ link = value; }
	void Start (int value)		{ start = value; }
	void End (int value)		{ end = value; }
	void Vol (int value)		{ vol = value; }
	void Cap (int value)		{ cap = value; }
	void Ratio (double value)		{ ratio = value; }
	void Oldtoll (int value)		{ oldtoll = value; }
	void Basetoll (int value)		 { basetoll = value; }
    void Newtoll(int value)		{ newtoll = value; }

private:
	int		link;
	int		start;
    int		end;
	int		vol;
    int		cap;
    double        ratio;
       int           oldtoll;
       int           basetoll;
    int           newtoll;

};

//--------------------------------------------------------
//      DToll_Output_Array class definition
//--------------------------------------------------------
class DToll_Output_Array : public Class_Array
{
public:
       DToll_Output_Array (int max_records = 0);

       bool Add (DToll_Output_Data *data = NULL)  { return (Class_Array::Add (data)); }

       DToll_Output_Data * New_Record (bool clear = false, int number = 1)
                                   { return ((DToll_Output_Data *)
Class_Array::New_Record (clear, number)); }

       DToll_Output_Data * Record (int index)     { return ((DToll_Output_Data *)
Class_Array::Record (index)); }
       DToll_Output_Data * Record (void)          { return ((DToll_Output_Data *)
Class_Array::Record ()); }

       DToll_Output_Data * Get (int index)         { return ((DToll_Output_Data *)
Class_Array::Get (index)); }
       DToll_Output_Data * Get_GE (int index)      { return ((DToll_Output_Data *)
Class_Array::Get_GE (index)); }
       DToll_Output_Data * Get_LE (int index)      { return ((DToll_Output_Data *)
Class_Array::Get_LE (index)); }

       DToll_Output_Data * First (void)           { return ((DToll_Output_Data *)
Class_Array::First ()); }
       DToll_Output_Data * Next (void)            { return ((DToll_Output_Data *)
Class_Array::Next ()); }

       DToll_Output_Data * Last (void)            { return ((DToll_Output_Data *)
Class_Array::Last ()); }
       DToll_Output_Data * Previous (void)        { return ((DToll_Output_Data *)
Class_Array::Previous ()); }

       DToll_Output_Data * First_Key (void)       { return ((DToll_Output_Data *)
Class_Array::First_Key ()); }
       DToll_Output_Data * Next_Key (void)        { return ((DToll_Output_Data *)
Class_Array::Next_Key ()); }
```

```
	DToll_Output_Data * operator[] (int index) { return (Record (index)); }
};

//---------------------------------------------------------
//	DToll_File Class definition
//---------------------------------------------------------
class  DToll_File : public Db_Header
{
public:
	DToll_File (Access_Type access = READ, Format_Type format = DEFAULT_FORMAT, bool
notes_flag = false);
	DToll_File (char *filename, Access_Type access = READ, Format_Type format =
DEFAULT_FORMAT, bool notes_flag = false);
	virtual ~DToll_File (void);

	int Index (void)		{ Get_Field (index, &lvalue); return (lvalue); }
	int Link (void)				{ Get_Field (link, &lvalue);
return (lvalue); }
	int Key (void)				{ Get_Field (key, &lvalue); return
(lvalue); }
	char *  Start (void)		{ Get_Field (start, svalue); return
(svalue); }
	char * End (void)			{ Get_Field (end, svalue); return
(svalue); }
	char * Use (void)			{ Get_Field (use, svalue); return
(svalue); }
	int Toll (void)				{ Get_Field (toll, &lvalue);
return (lvalue); }
    int Marker (void)			{ Get_Field (marker, &lvalue); return (lvalue);}
// to read new marker toll file
    int Node (void)				{ Get_Field (node, &lvalue); return
(lvalue);}  // to read new node toll file
    char* Notes (void)			{ Get_Field (notes, svalue); return
(svalue);}  // to read new node toll file

	void Index(int value)		{ Put_Field (index, value); }
	void Link (int value)		{ Put_Field (link, value); }
	void Key (int value)		{ Put_Field (key, value); }
	void Start (char* value)	{ Put_Field (start, value); }
	void End (char* value)			{ Put_Field (end, value); }
	void Use (char* value)			{ Put_Field (use, value); }
	void Toll (int value)		{ Put_Field (toll, value); }
    void Marker (int value)		{ Put_Field (marker, value); } // to write
new marker toll file
    void Node (int value)		{ Put_Field (node, value);}    // to read
new node toll file
```

```
    void Notes (char* value)        { Put_Field (notes, value);}   // to read new node
toll file

       bool Use_Flag (void)               { return (use != 0); }

       virtual bool Create_Fields (void);

protected:
       virtual bool Set_Field_Numbers (void);

private:
       void Setup (void);

       int lvalue;

       int key, link, dir, start, end, use, marker, toll, node, notes, index; // for
putting marker option in the toll file
};

//-----------------------------------------------------------
//     Lookuptable_Data class definition
//-----------------------------------------------------------
class Lookuptable_Data : public Class_Index
{
public:
       Lookuptable_Data (int index = 0);
       virtual ~Lookuptable_Data (void)     {}

       int     Index(void)                  { return (ID ()); }
       double  Lowvc (void)                 { return (lowvc); }
       double  Highvc (void)                { return (highvc); }
       double  Toll (void)                  { return (toll); }

       void Index  (int value)            { ID (value); }
       void Lowvc  (double value)         { lowvc = value; }
       void Highvc (double value)         { highvc = value; }
       void Toll   (double value)         { toll = value; }

protected:
       void Copy_Data (char * value);

private:
       double lowvc;
       double highvc;
       double toll;
};
```

```
//-----------------------------------------------------------
//      Lookuptable_Array class definition
//-----------------------------------------------------------
class Lookuptable_Array : public Class_Array
{
public:
        Lookuptable_Array (int max_records = 0);

        bool Add (Lookuptable_Data *data = NULL)  { return (Class_Array::Add (data)); }

        Lookuptable_Data * New_Record (bool clear = false, int number = 1)
                                   { return ((Lookuptable_Data *)
Class_Array::New_Record (clear, number)); }

        Lookuptable_Data * Record (int index)     { return ((Lookuptable_Data *)
Class_Array::Record (index)); }
        Lookuptable_Data * Record (void)          { return ((Lookuptable_Data *)
Class_Array::Record ()); }

        Lookuptable_Data * Get (int index)         { return ((Lookuptable_Data *)
Class_Array::Get (index)); }
        Lookuptable_Data * Get_GE (int index)      { return ((Lookuptable_Data *)
Class_Array::Get_GE (index)); }
        Lookuptable_Data * Get_LE (int index)      { return ((Lookuptable_Data *)
Class_Array::Get_LE (index)); }

        Lookuptable_Data * First (void)           { return ((Lookuptable_Data *)
Class_Array::First ()); }
        Lookuptable_Data * Next (void)            { return ((Lookuptable_Data *)
Class_Array::Next ()); }

        Lookuptable_Data * Last (void)            { return ((Lookuptable_Data *)
Class_Array::Last ()); }
        Lookuptable_Data * Previous (void)        { return ((Lookuptable_Data *)
Class_Array::Previous ()); }

        Lookuptable_Data * First_Key (void)       { return ((Lookuptable_Data *)
Class_Array::First_Key ()); }
        Lookuptable_Data * Next_Key (void)        { return ((Lookuptable_Data *)
Class_Array::Next_Key ()); }

        Lookuptable_Data * operator[] (int index) { return (Record (index)); }
};

//-----------------------------------------------------------
//      Lookuptable_File Class definition
//-----------------------------------------------------------
```

```
class  Lookuptable_File : public Db_Header
{
public:

      Lookuptable_File (Access_Type access = READ, Format_Type format = DEFAULT_FORMAT,
bool notes_flag = false);
       Lookuptable_File (char *filename, Access_Type access = READ, Format_Type format
= DEFAULT_FORMAT, bool notes_flag = false);
       virtual ~Lookuptable_File (void);

       int Index (void)                          { Get_Field (index, &value); return
(value); }
       double Lowvc (void)                       { Get_Field (lowvc, &lvalue); return
(lvalue); }
       double Highvc (void)               { Get_Field (highvc, &lvalue); return
(lvalue); }
       double Toll (void)                        { Get_Field (toll, &lvalue); return
(lvalue); }

       void Index (int value)                    { Put_Field (index, value); }
       void Lowvc (double value)          { Put_Field (lowvc, value); }
       void Highvc (double value)         { Put_Field (highvc, value); }
       void Toll (double value)           { Put_Field (toll, value); }

    //bool Use_Flag (void)                { return (use != 0); }

       virtual bool Create_Fields (void);

protected:
       virtual bool Set_Field_Numbers (void);

private:
       void Setup (void);

       double lvalue;
       int    value;

       int index, lowvc, highvc, toll; // for putting marker option in the toll file
};

#endif
```

Appendix C. Added/Modified Source Codes for Departure Time Choice Model

< /Departselect/Read_Plan.cpp >

```
//*********************************************************
//      Read_Plan.cpp - Read Each Plan File
//*********************************************************
#include "DepartSelect.hpp"
#include "In_Polygon.hpp"
#include "Utility.hpp"

//---------------------------------------------------------
//      Read_Plan
//---------------------------------------------------------
void DepartSelect::Read_Plan (void)
{
        int hhold, age, current_hhold, person, num_sel, after_dtc_num_sel,
               before_dtc_num_sel, cap, offset, dir, list, tod_cap, lanes,is_changed =
0 ;
        int id, n1, n2, num, time, num_path, *path, skim, ttime, SD, ratio,
               min_vc, in_link_dir,deltaTR,diff;

        int num_in, num_out, vehicle, activitynumber, startinglocation,
               endinglocation, current_dtc_record_num, num_new_activity_record;

        bool select, gender, first, traveler_flag, location_flag, path_flag,
               time_flag, park_lot_flag, stop_flag;

        bool org_flag, des_flag, mid_flag, ie_flag, ncoord_flag, nin_flag, nout_flag;
        double share, cap_factor, deltaDT, IBD, deltaT, uoa, LMD;
        Box *coord_ptr, *in_ptr, *out_ptr;

        Uoa_data *utility = NULL;

        Population_Data *population_ptr;
        AB_Data *ab_ptr;
        TTime_Data *ttime_ptr;
        Link_Data *link_ptr;
        Parking_Data *parking_ptr;
        Link_Use_Data *use_ptr;
        Connect_Time *connect_ptr;
        Node_Data *node_ptr;
        Vehicle_Data *vehicle_ptr;

        ie_flag = Report_Flag (SUBAREA_IO);

        if (coord_flag && coord_box.Num_Records () == 1) {
```

```
			ncoord_flag = false;
			coord_ptr = (Box *) coord_box.First ();
		}  else {
			ncoord_flag = true;
		}
		if (in_flag && od_box.Num_Records () == 1) {
			nin_flag = false;
			in_ptr = (Box *) od_box.First ();
		}  else {
			nin_flag = true;
		}
		if (out_flag && exclude_box.Num_Records () == 1) {
			nout_flag = false;
			out_ptr = (Box *) exclude_box.First ();
		}  else {
			nout_flag = true;
		}

		//---- set the filename and extension ----

		if (plan_file.Extend ()) {
			Show_Message ("Reading Plan File %s -- Plan", plan_file.Extension ());
			//str_fmt (filename, sizeof (filename), "%s.%s", hhold_name,
plan_file.Extension ());
		} else {
			Show_Message ("Reading Plan File -- Plan");
			//str_cpy (filename, sizeof (filename), hhold_name);
		}
		Set_Progress (10000);

		//---- open the after departure choice household id file ----

		if(use_of_after_dtc_holdlistfile_flag) {
		after_dtc_hhold_file = f_open (after_dtc_hhold_name, "wt");

		if (after_dtc_hhold_file == NULL) {
			File_Error ("Creating after dtc Household ID File", after_dtc_hhold_name);
		}
		}

		//---- open the pre departure choice household id file ----

		before_dtc_hhold_file = f_open (before_dtc_hhold_name, "wt");

		if (before_dtc_hhold_file == NULL) {
			File_Error ("Creating before dtc Household ID File",
before_dtc_hhold_name);
```

```
	}

	//---- open the new activity file ----

	if (new_activity_file_flag){
	new_activity_file = f_open (new_activity_file_name, "wt");

	if (new_activity_file == NULL) {
		File_Error ("Creating bnew activity File", new_activity_file_name);
	}
	else{
		Write_output_new_activityfile_header(new_activity_file);
		Setup_new_activity_array ();
	    }
	}

	if(activity_file_hour_flag)
		 Activity_time_transform_H_S();

	//---- open the current departure choice output file ----

	current_dtc_output_file = f_open (Departchoice_output_name, "wt");

	if (current_dtc_output_file == NULL) {
		File_Error ("Creating current departure choice output File",
Departchoice_output_name);
	}
	else{
		Write_output_file_header (current_dtc_output_file);
	}

	hhold_list.Reset ();
	after_dtc_hhold_list.Reset ();
	before_dtc_hhold_list.Reset ();

	//---- read each plan record ----

	current_hhold = num_hh = num_sel = after_dtc_num_sel = before_dtc_num_sel =
current_dtc_record_num = num_new_activity_record = 0;
	traveler_flag = (travelers.Num_Ranges () > 0);
	time_flag = (times.Num_Ranges () > 0);
	location_flag = (locations.Num_Ranges () > 0);
	park_lot_flag = (parking_lots.Num_Ranges () > 0);
	stop_flag = (transit_stops.Num_Ranges () > 0);
	path_flag = (nodes.Num_Records () > 0);

	Uoa_data *utility_ptr = new Uoa_data;
	if (vc_flag) {
		num = ttime_data.Periods ();
		if (num > 0) {
			cap_factor = 24.0 / num;
		}
		min_vc = (int) (vc_ratio * 1000.0 + 0.5);
	}

	//Get_totalpopulation_in_activity ();

	total_eligible_people = 0;

	while (plan_file.Read ()) {
		Show_Progress ();

		//---- get the traveler id ----

		person = plan_file.Person ();
		hhold  = plan_file.Household ();
		startinglocation = plan_file.Start_ID();
		endinglocation = plan_file.End_ID();

		//---- convert the traveler id to household id ----

		if (pop_flag) {
			population_ptr = population_data.Get (hhold,person);
			if (population_ptr == NULL) {
				Error ("Traveler %d was Not Found in the Population File",
person);
			}
			//hhold = population_ptr->Household();
			age    = population_ptr->Age();
			gender  = population_ptr->Gender();

		} else {
			hhold = plan_file.Traveler() / Traveler_Scale ();
		}
		if (hhold <= 0) {
			Error ("Household ID %d is Out of Range", hhold);
		}
		if (current_hhold != hhold) {
			current_hhold = hhold;
			num_hh++;
		}

		//---- check for travel activities ----
```

```
if (plan_file.Start_Type () == LOCATION_ID &&
        plan_file.End_Type () == LOCATION_ID) continue;

//---- check the traveler selection criteria ----

if (traveler_flag) {
        if (!travelers.In_Range (plan_file.Traveler())) continue;
}

//---- check the vehicle type selection criteria ----

if (vehicle_flag) {
        vehicle = plan_file.Vehicle ();
        if (vehicle <= 0) continue;

        vehicle_ptr = vehicle_data.Get (vehicle);
        if (vehicle_ptr == NULL) continue;

        if (!vehicle_types.In_Range (vehicle_ptr->Type ())) continue;
}

//---- check the time period criteria ----

if (time_flag) {

        time = plan_file.Time ();

        //---- apply the time scaling factor ----

        if (!times.In_Range (time)) continue;
}

//---- check the activity location criteria ----

if (location_flag) {
        num = 0;
        if (plan_file.Start_Type () == LOCATION_ID) {
                if (locations.In_Range (plan_file.Start_ID ())) {
                        num++;
                }
        }
        if (plan_file.End_Type () == LOCATION_ID) {
                if (locations.In_Range (plan_file.End_ID ())) {
                        num++;
                }
        }
```

```
		if (!num) continue;
	}

	//---- check the parking lot criteria ----

	if (park_lot_flag) {
		num = 0;
		if (plan_file.Start_Type () == PARKING_ID) {
			if (parking_lots.In_Range (plan_file.Start_ID ())) {
				num++;
			}
		}
		if (plan_file.End_Type () == PARKING_ID) {
			if (parking_lots.In_Range (plan_file.End_ID ())) {
				num++;
			}
		}
		if (!num) continue;
	}

	//---- check the transit stop criteria ----

	if (stop_flag) {
		num = 0;
		if (plan_file.Start_Type () == STOP_ID) {
			if (transit_stops.In_Range (plan_file.Start_ID ())) {
				num++;
			}
		}
		if (plan_file.End_Type () == STOP_ID) {
			if (transit_stops.In_Range (plan_file.End_ID ())) {
				num++;
			}
		}
		if (!num) continue;
	}

	//---- od coordinate exclusion ----

	if (in_flag || out_flag) {
		path = plan_file.Path (&num_path);

		if (num_path > 0) {
			num_in = num_out = 0;

			for (n1=0; n1 < 2; n1++) {
				id = (n1 == 0) ? *path : path [num_path-1];
				//---- link list path ----

				if (!type_flag) {
					if (id < 0) {
						dir = 1;
						id = -id;
					} else {
						dir = 0;
					}
					link_ptr = link_data.Get (id);
					if (link_ptr == NULL) continue;

					if (dir == 0) {
						id = link_ptr->Bnode ();
					} else {
						id = link_ptr->Anode ();
					}
				}

				//---- check the node coordinate ----

				node_ptr = node_data.Get (id);
				if (node_ptr == NULL) continue;

				if (in_flag) {
					if (nin_flag) {
						for (in_ptr = (Box *) od_box.First ();
in_ptr; in_ptr = (Box *) od_box.Next ()) {
								if (node_ptr->X () >= in_ptr-
>x1 && node_ptr->X () <= in_ptr->x2 &&
									node_ptr->Y () >=
in_ptr->y1 && node_ptr->Y () <= in_ptr->y2) num_in++;
						}
					} else {
						if (node_ptr->X () >= in_ptr->x1 &&
node_ptr->X () <= in_ptr->x2 &&
							node_ptr->Y () >= in_ptr->y1
&& node_ptr->Y () <= in_ptr->y2) num_in++;
					}
				}
				if (out_flag) {
					if (nout_flag) {
						for (out_ptr = (Box *)
exclude_box.First (); out_ptr; out_ptr = (Box *) exclude_box.Next ()) {
								if (node_ptr->X () < out_ptr-
>x1 || node_ptr->X () > out_ptr->x2 ||
                                                                        node_ptr->Y () <
out_ptr->y1 || node_ptr->Y () > out_ptr->y2) num_out++;
                                                        }
                                                } else {
                                                        if (node_ptr->X () < out_ptr->x1 ||
node_ptr->X () > out_ptr->x2 ||
                                                                node_ptr->Y () < out_ptr->y1
|| node_ptr->Y () > out_ptr->y2) num_out++;
                                                }
                                        }
                                }
                                if (in_flag && num_in < 1) continue;
                                if (out_flag && num_out < 2) continue;
                        }
                }

                //---- coordinate selection ----

                if (coord_flag) {
                        path = plan_file.Path (&num_path);

                        while (num_path-- > 0) {
                                id = *path++;

                                //---- link list path ----

                                if (!type_flag) {
                                        if (id < 0) {
                                                dir = 1;
                                                id = -id;
                                        } else {
                                                dir = 0;
                                        }
                                        link_ptr = link_data.Get (id);
                                        if (link_ptr == NULL) continue;

                                        if (dir == 0) {
                                                id = link_ptr->Bnode ();
                                        } else {
                                                id = link_ptr->Anode ();
                                        }
                                }

                                //---- check the node coordinate ----

                                node_ptr = node_data.Get (id);
                                if (node_ptr == NULL) continue;
```

```
					if (ncoord_flag) {
						for (coord_ptr = (Box *) coord_box.First ();
coord_ptr; coord_ptr = (Box *) coord_box.Next ()) {
							if (node_ptr->X () >= coord_ptr->x1 &&
node_ptr->X () <= coord_ptr->x2 &&
								node_ptr->Y () >= coord_ptr->y1 &&
node_ptr->Y () <= coord_ptr->y2) break;
						}
						if (coord_ptr != NULL) break;
					} else {
						if (node_ptr->X () >= coord_ptr->x1 && node_ptr->X
() <= coord_ptr->x2 &&
							node_ptr->Y () >= coord_ptr->y1 && node_ptr-
>Y () <= coord_ptr->y2) break;
					}
				}
				if (num_path < 0) continue;
			}

			//---- check the subarea ----

			if (subarea_flag) {
				path = plan_file.Path (&num_path);
				org_flag = des_flag = mid_flag = select = false;
				first = true;

				while (num_path-- > 0) {
					id = *path++;

					//---- link list path ----

					if (!type_flag) {
						if (id < 0) {
							dir = 1;
							id = -id;
						} else {
							dir = 0;
						}
						link_ptr = link_data.Get (id);
						if (link_ptr == NULL) continue;

						if (dir == 0) {
							id = link_ptr->Bnode ();
						} else {
							id = link_ptr->Anode ();
						}
```

```
                                        }

                                        //---- check the node coordinate ----

                                        node_ptr = node_data.Get (id);
                                        if (node_ptr == NULL) continue;

                                        select = In_Polygon (UnRound (node_ptr->X ()), UnRound
(node_ptr->Y ()), &select_subarea.points);

                                        if (ie_flag) {
                                                if (select) mid_flag = true;
                                                if (first) {
                                                        org_flag = select;
                                                        first = false;
                                                }
                                                des_flag = select;
                                        } else {
                                                if (select) break;
                                        }
                                }
                                if (ie_flag) {
                                        if (!mid_flag) continue;
                                        if (org_flag) {
                                                if (des_flag) {
                                                        num_ii++;
                                                } else {
                                                        num_ie++;
                                                }
                                        } else {
                                                if (des_flag) {
                                                        num_ei++;
                                                } else {
                                                        num_ee++;
                                                }
                                        }
                                } else if (num_path < 0) {
                                        continue;
                                }
                        }

                        //---- check the nodes groups ----

                        if (path_flag) {
                                select = false;
                                Data_Range *range;
```

```
                    for (range = (Data_Range *) nodes.First (); range != NULL
&& !select; range = (Data_Range *) nodes.Next ()) {

                            path = plan_file.Path (&num_path);
                            num = range->Num_Ranges ();
                            n1 = n2 = 0;

                            while (num_path-- > 0) {
                                    id = *path++;

                                    //---- link list path ----

                                    if (!type_flag) {
                                            if (id < 0) {
                                                    dir = 1;
                                                    id = -id;
                                            } else {
                                                    dir = 0;
                                            }
                                            link_ptr = link_data.Get (id);
                                            if (link_ptr == NULL) continue;

                                            if (dir == 0) {
                                                    id = link_ptr->Bnode ();
                                            } else {
                                                    id = link_ptr->Anode ();
                                            }
                                    }

                                    //---- check the node sequence ----

                                    n2 = range->In_Index (id);

                                    if (!n2) continue;

                                    if (n2 != n1 + 1) {
                                            n2 = 0;
                                            break;
                                    }
                                    if (n2 == num) {
                                            select = true;
                                            break;
                                    }
                                    n1 = n2;
                            }
                    }
                    if (!select) continue;
```

```
		}

		//---- apply the path-based selection criteria to drive legs ----

		if (vc_flag || diff_flag || ratio_flag) {
			if (plan_file.Start_Type () != PARKING_ID ||
				plan_file.End_Type () != PARKING_ID) continue;

			time = Round (plan_file.Time ());

			skim = n1 = in_link_dir = 0;
			path = plan_file.Path (&num_path);
			first = true;

			//---- get the travel time from the origin parking lot ----

			if (parking_flag) {
				parking_ptr = parking_data.Get (plan_file.Start_ID ());

				if (parking_ptr != NULL) {
					link_ptr = link_data.Get (parking_ptr->Link ());

					if (link_ptr != NULL) {
						if ((type_flag && *path == link_ptr->Anode
()) || *path < 0) {
							offset = parking_ptr->Offset ();
							dir = link_ptr->BA_Dir ();
						} else {
							offset = link_ptr->Length () -
parking_ptr->Offset ();

							dir = link_ptr->AB_Dir ();
						}
						if (dir > 0) {
							ttime_ptr = ttime_data [dir];

							num = ttime_data.Period (Resolve
(time));


							ttime = ttime_ptr->TTime (num) *
offset / link_ptr->Length ();


							skim += ttime;
							time += ttime;
							first = false;

							in_link_dir = ttime_ptr->Link_Dir ();
						}
```

```
                }
            }
        }

        //---- re-skim the path with current travel times ----

        while (num_path-- > 0) {
            n2 = *path++;

            if (n1 != 0) {

                //---- get the link direction ----

                if (type_flag) {
                    ab_ptr = ab_key.Get (n1, n2);

                    if (ab_ptr == NULL) {
                        Error ("Anode-Bnode %d-%d was Not
Found in the Link File", n1, n2);
                    }
                    dir = ab_ptr->dir;
                } else {
                    link_ptr = link_data.Get (abs (n2));

                    if (link_ptr == NULL) {
                        Error ("Link %d was Not Found in the
Link File", abs (n2));
                    }
                    if (n2 < 0) {
                        dir = link_ptr->BA_Dir ();
                    } else {
                        dir = link_ptr->AB_Dir ();
                    }
                }
                if (dir == 0) {
                    n1 = n2;
                    continue;
                }

                //---- get the link travel time ----

                ttime_ptr = ttime_data [dir];

                num = ttime_data.Period (Resolve (time));

                ttime = ttime_ptr->TTime (num);
```

```
                            //---- check the time ratio ----

                            if (ratio_flag) {
                                share = (double) ttime / ttime_ptr->Time0 ();
                                 //if (share > time_ratio) goto keep;
                            }

                            //---- add turning movement delay ----

                            if (turn_flag) {
                                 connect_ptr = connect_time.Get (in_link_dir,
ttime_ptr->Link_Dir ());

                                 in_link_dir = ttime_ptr->Link_Dir ();

                                 if (connect_ptr != NULL && connect_ptr-
>Periods () > 0) {

                                      ttime += connect_ptr->TTime (num);
                                      if (ttime < 1) ttime = 1;
                                 }
                            }

                            //---- check the V/C ratio ----

                            if (vc_flag) {
                                 cap = (int) (ttime_ptr->Capacity () *
cap_factor + 0.5);


                                 if (cap > 0) {
                                      list = ttime_ptr->TOD_List ();

                                      for (; list > 0; list = use_ptr-
>TOD_List ()) {

                                           use_ptr = link_use_data [list];


                                            if (use_ptr->Start () <= time
&& time < use_ptr->End ()) {

                                                 lanes = ttime_ptr->Thru
();

                                                 if (lanes < 1) lanes =
1;


                                                 tod_cap = (cap *
use_ptr->Lanes () + lanes / 2) / lanes;

                                                 if (tod_cap == 0)
tod_cap = cap / 2;

                                                 if (tod_cap < 1)
tod_cap = 1;
```

```
                                                            cap = tod_cap;
                                                            break;
                                                    }
                                            }
                                            ratio = (1000 * ttime_ptr->Volume
(num) + (cap >> 1)) / cap;

                                            //if (ratio > min_vc) goto keep;
                                    }
                            }
                            if (first) {
                                    first = false;
                                    ttime += (ttime >> 1);
                            }
                            skim += ttime;
                            time += ttime;
                    }
                    n1 = n2;
            }

            //---- travel time difference ----

            if (diff_flag) {

                    //---- get the travel time to the destination parking lot -
---

                    if (parking_flag) {
                            parking_ptr = parking_data.Get (plan_file.End_ID
());

                            if (parking_ptr != NULL) {
                                    link_ptr = link_data.Get (parking_ptr->Link
());

                                    if (link_ptr != NULL) {
                                            if (type_flag) {
                                                    if (n1 == link_ptr->Anode ())
{
                                                            offset = parking_ptr-
>Offset ();
                                                            dir = link_ptr->AB_Dir
();
                                                    } else {
                                                            offset = link_ptr-
>Length () - parking_ptr->Offset ();
                                                            dir = link_ptr->BA_Dir
();
```

```
                                                  }
                                                  if (dir > 0) {

                                                          //---- get the link
travel time ----

                                                          ttime_ptr = ttime_data
[dir];

                                                          num = ttime_data.Period
(Resolve (time));

                                                          ttime = ttime_ptr-
>TTime (num) * offset / link_ptr->Length ();

                                                          //---- add turning
movement delay ----

                                                          if (turn_flag) {
                                                                  connect_ptr =
connect_time.Get (in_link_dir, ttime_ptr->Link_Dir ());

                                                                  if
(connect_ptr != NULL && connect_ptr->Periods () > 0) {
                                                                          ttime +=
connect_ptr->TTime (num);
                                                                          if
(ttime < 1) ttime = 1;
                                                                  }
                                                          }
                                                  }

                                          } else {

                                                  skim -= ttime;
                                                  time -= ttime;

                                                  if (n1 > 0) {
                                                          offset = parking_ptr-
>Offset ();
                                                  } else {
                                                          offset = link_ptr-
>Length () - parking_ptr->Offset ();
                                                  }
                                                  ttime = ttime * offset /
link_ptr->Length ();
                                          }
```

```
                                                skim += ttime;
                                                time += ttime;
                                                first = true;
                                        }
                                }
                        }
                        if (!first) {
                                skim += (ttime >> 1);
                        }
                        skim = Resolve (skim);

                        //---- compare to the plan duration ----

                        time = plan_file.Duration ();

                        //---- check the selection criteria ----

                        SD = skim - time;

                        diff = abs (time - skim);

                        if (diff < min_time) continue;

                        if (diff < max_time) {
                                if (time > 0) {
                                        share = (double) diff / (double) time;
                                } else {
                                        share = 1.0;
                                }
                                if (share < percent_diff) continue;
                        }
                } else {
                        continue;
                }


                    activitynumber = Get_activitivityinfor_FAST (startinglocation,
endinglocation, 0 ,  hhold,  person);

                //if(activity_file_hour_flag)
                        //      Activity_time_transform_H_S_FAST( hhold,  person,
activitynumber);

                        deltaTR = DeltaTR_FAST (num_iteration, skim, gender, hhold,
person, activitynumber);

                        deltaDT = DeltaDT_FAST (num_iteration, gender, hhold,
person, activitynumber);

                        LMD = Cal_LMD_FAST (num_iteration, gender, hhold, person,
activitynumber);

                        IBD = IBD_Cal (SD, age, gender, deltaTR, deltaDT, LMD);

                        deltaT = Cal_deltaT (SD, IBD);

                       Alternative_Cal (hhold, person, activitynumber,utility_ptr);

                        Output_data* datatmp = new Output_data;

           datatmp->hhold  =  hhold;
                        datatmp->person =  person;
                        datatmp->age =  age;
           datatmp->gender =  gender;
                        datatmp->activity =  activitynumber;
                        datatmp->Et =  skim;
                        datatmp->Rt =  time;
           datatmp->Sd =  SD;
           datatmp->Dtr = deltaTR;
           datatmp->Ddt = deltaDT;
           datatmp->Ibd = IBD;
                        datatmp->utility = utility_ptr->uoa;
                        uoa = utility_ptr->uoa;
           datatmp->dt = deltaT;
           datatmp->flexibility = utility_ptr->flexibility;
           datatmp->retire = utility_ptr->retire;
           datatmp->flextimework = utility_ptr->flextimework;
                        datatmp->income =  utility_ptr->income;
           datatmp->trippurpose = utility_ptr->trippurpose;

                        if(Is_eligible (SD, IBD, hhold, person, activitynumber) &&
uoa > 0.0 &&  Is_eligibletraveler_C_1 (SD, IBD))
                                is_changed = 1;
                        else
               is_changed = 0;

                        datatmp->is_changed = is_changed;

                        Write_output_file (current_dtc_output_file, datatmp);

                        delete datatmp;

                        if(Is_eligibletraveler_C_1 (SD, IBD)){
                                current_dtc_record_num++;
```

```
                                        if (!before_dtc_hhold_list.Get_Index (hhold)) {
                                                before_dtc_num_sel++;
                                                if (!before_dtc_hhold_list.Add (hhold)) {
                                                Error ("Adding Household to the List");
                                                }
                                        }
                                }

                                if(use_of_dtc_flag){
                                if(Is_eligibletraveler_C_1 (SD, IBD) && Is_eligible (SD,
IBD, hhold, person, activitynumber) && uoa > 0.0 ){
                                        if(new_activity_file_flag){
                        num_new_activity_record++;
                                        Change_activity_ptr_FAST (hhold, person,
activitynumber, deltaT);
                                        }
                                        if(use_of_after_dtc_holdlistfile_flag){
                                                if (!after_dtc_hhold_list.Get_Index (hhold))
{
                                                        after_dtc_num_sel++;
                                                       if (!after_dtc_hhold_list.Add (hhold))
{
                                                                Error ("Adding Household to
the List");
                                                        }
                                                }
                                        }
                                 }
                                }
                                //if(Is_stop(SD,IBD, hhold, person, activitynumber))
                                //      break;

                }

                //---- add the household to the list ----
keep:
        if (!hhold_list.Get_Index (hhold)) {
                        num_sel++;
                        if (!hhold_list.Add (hhold)) {
                                Error ("Adding Household to the List");
                        }
                }
        }
        End_Progress ();

        Print (2, "Plan File: %s", plan_file.Filename ());
        Print (1, "Total Number of Households = %d", num_hh);
        Print (2, "₩tstarting to write before_dtc_hhold_file");
        Write_output_hhold_list (before_dtc_hhold_file, before_dtc_num_sel,
before_dtc_hhold_list );

        if(use_of_after_dtc_holdlistfile_flag){
                Print (2, "₩tstarting to write after_dtc_hhold_file");
                Write_output_hhold_list (after_dtc_hhold_file , after_dtc_num_sel,
after_dtc_hhold_list);
        }
        if(new_activity_file_flag){
                if(activity_file_hour_flag)
                        Activity_time_transform_S_H ();
                Write_output_new_activityfile (new_activity_file);
        }

        Print (2, "The number of new activityfile records changed are = %d",
                num_new_activity_record);

        Print (2, "Number of current_dtc_record_number written = %d",
current_dtc_record_num);

        Close_files ();

}
```

< /Departselect/Alternative_Cal.cpp >

```
//***********************************************************
//     Alternative_Cal.cpp - calculate the utility of alternative
//***********************************************************
#include "DepartSelect.hpp"
#include "In_Polygon.hpp"
#include "Utility.hpp"

#define constant  -0.66
#define constant_flexibility 0.71
#define constant_retire 1.1
#define constant_flex_timework 0.93
#define constant_income -0.74
#define constant_trip_purpose -0.48

//-----------------------------------------------------------
//     Alternative_Cal
//-----------------------------------------------------------
void DepartSelect::Alternative_Cal (int hhold, int person, int activitynumber, Uoa_data
*uoa_ptr)
{
       double uoa;
       int flexibility,flextimework, trippurpose,retire,income, index = 0;

       //Get_Randomnumber_FAST(hhold, person, activitynumber, flexibility,
flextimework );
       index++;
       if(index<1000000){
       flexibility = 1; //Binary_number_gen();
       flextimework = 1; //Nonuniform_binary_number_generator();
       }
       income = Income_FAST(hhold,person);
    trippurpose = trip_purpose(hhold,person,activitynumber);
    retire = is_retire_FAST(hhold,person);

       uoa = constant+ constant_flexibility*flexibility+constant_retire*retire+
              constant_flex_timework*flextimework+constant_income*income
              +constant_trip_purpose*trippurpose;

       uoa_ptr->flexibility = flexibility;
    uoa_ptr->flextimework = flextimework;
    uoa_ptr->income = income;
    uoa_ptr->trippurpose = trippurpose;
    uoa_ptr->retire = retire;
    uoa_ptr->uoa = uoa;
}
```

```
//-----------------------------------------------------------
//      is_retire
//-----------------------------------------------------------
bool DepartSelect::is_retire (int hhold, int person)
{
        bool retire = false, found = false;

        Population_Data *population_ptr;
    population_ptr = population_data.First();
        if (population_ptr->Household() == hhold && population_ptr->Person() == person){
                if(population_ptr->Age()>=65)
                        retire = true;
                else
                        retire = false;
                return retire;
        }

        while((population_ptr=population_data.Next())!=NULL){
                if (population_ptr->Household() == hhold && population_ptr->Person() ==
person){
                        if(population_ptr->Age()>=65)
                                retire = true;
                        else
                                retire = false;

                        found = true;
                        break;
                }
        }
    population_ptr=population_data.Last();
        if (population_ptr->Household() == hhold && population_ptr->Person() == person){
                if(population_ptr->Age()>=65)
                        retire = true;
                else
                        retire = false;
                found = true;
        }
        if(!found)
                        Error ("is_retire, hhold %d, and %d person were not found in
population data", hhold, person);

        return retire;
}

//-----------------------------------------------------------
//      is_retire_FAST
//-----------------------------------------------------------
```

```
bool DepartSelect::is_retire_FAST(int hhold, int person)
{
    bool retire = false;
       Population_Data* population_ptr = NULL;

       population_ptr = population_data.Get(hhold, person);

       if(population_ptr == NULL){
                Print (2, "did not sucessfully transformed activity time (from hours to
seconds)for hhold %d, person %d", hhold, person);
             return false;
       }
       if(population_ptr->Age()>=65)
                      retire = true;
       else
                  retire = false;

       return retire;

}

//-----------------------------------------------------------
//      income
//-----------------------------------------------------------
int DepartSelect::Income (int hhold, int person)
{
       bool retire = false, found = false;
       int income = 0, incomethreshold = 75000;

       Population_Data *population_ptr;
    population_ptr = population_data.First();
       if (population_ptr->Household() == hhold && population_ptr->Person() == person){
              income = population_ptr->Income();
              if(income>=incomethreshold)
                     income = 1;
           else
                     income = 0;
              return income;
       }

       while((population_ptr=population_data.Next())!=NULL){
              if (population_ptr->Household() == hhold && population_ptr->Person() ==
person){
                     income = population_ptr->Income();
                     found = true;
                     break;
                }
        }
    population_ptr=population_data.Last();
        if (population_ptr->Household() == hhold && population_ptr->Person() == person){
                        income = population_ptr->Income();
                        found = true;
        }
        if(!found)
                        Error ("is_retire, hhold %d, and %d person were not found in
population data", hhold, person);

        if(income>=incomethreshold)
                income = 1;
        else
                income = 0;
        return income;
}

//--------------------------------------------------------------
//      Income_FAST
//--------------------------------------------------------------
bool DepartSelect::Income_FAST(int hhold, int person)
{
    int income = 0, incomethreshold = 75000;
        Population_Data* population_ptr = NULL;
        population_ptr = population_data.Get(hhold, person);

        if(population_ptr == NULL){
                  Print (2,"did not sucessfully transformed activity time (from hours to
seconds)for hhold %d, person %d", hhold, person);
              return 0;
        }
        income = population_ptr->Income();

        if(income>=incomethreshold)
                income = 1;
        else
                income = 0;
        return income;

}
```

< /Departselect/functionsets.cpp >

```
//**********************************************************
//     Utility.cpp - process the utility functions
//**********************************************************
#include "DepartSelect.hpp"
#include "In_Polygon.hpp"
#include "Utility.hpp"

//----------------------------------------------------------
//     DeltaTR
//----------------------------------------------------------
int DepartSelect::DeltaTR (int num_iteration, int ET, bool gender, int hhold, int
person, int activitynumber)
{
       int deltaTR = 0; int p_ET = 0, p_hhold, p_person; bool p_record_find = false;
       if (num_iteration == 1)
              return deltaTR;
       else{
              if(previous_output_flag){
              previous_departchoice_file.Rewind();
              while (previous_departchoice_file.Read ()) {
                     //Show_Progress ();
                     p_person = previous_departchoice_file.Person ();
                     p_hhold  = previous_departchoice_file.Hhold ();

                     if (hhold==p_hhold && p_person==person && activitynumber ==
previous_departchoice_file.Activity()){
                            p_ET=previous_departchoice_file.Et();
                            p_record_find=true;
                            break;
                     }
              }
              if (!p_record_find){
                     Print (2, "DeltaTR:Household ID %d, person ID %d, and acitivity ID
%d was not found in the pervious_output file", hhold, person,activitynumber);
                  return deltaTR;
              }

              if (num_iteration>1 && !previous_output_flag)
                     Error ("DeltaTR:This is the number of iteration %d, which is
greater than 1, you should specify one pervious output file ", num_iteration);

              deltaTR=ET-p_ET;
              return deltaTR;
              }
              else{
```

```
            Error ("DeltaTR:Household ID %d, person ID %d, and acitivity ID %d was
not found in the pervious_output file", hhold, person,activitynumber);
            }
        }
}

//----------------------------------------------------------
//      DeltaTR_FAST
//----------------------------------------------------------
int DepartSelect::DeltaTR_FAST(int num_iteration, int ET, bool gender, int hhold, int
person, int activitynumber)
{
        int deltaTR = 0; int p_ET = 0;
    Departchoice_Data* Depart_ptr = NULL;

        if (num_iteration>1 && !previous_output_flag)
            Error ("DeltaTR:This is the number of iteration %d, which is greater than
1, you should specify one pervious output file ", num_iteration);

        if (num_iteration == 1)
            return deltaTR;
        else{
            if(previous_output_flag){
                Depart_ptr = pre_departchoice_data.Get(hhold*Traveler_Scale () +
person,activitynumber);
                if(Depart_ptr == NULL){
                    Print (2, "this is not a valid record, we can not solve the
DeltaTR for hhold %d, person %d, activity number %d" ,hhold, person, activitynumber);
                    return deltaTR;
                }
                p_ET=Depart_ptr->Et();
            }
        }
        deltaTR=ET-p_ET;
        return deltaTR;

}

//----------------------------------------------------------
//      DeltaDT
//----------------------------------------------------------
double DepartSelect::DeltaDT (int num_iteration, bool gender, int hhold, int person,
int activitynumber)
{
        double deltaDT = 1; double p_DT = 0, p_p_DT = 0; int p_hhold, p_person,
p_p_hhold, p_p_person; bool p_record_find = false, p_p_record_find = false;
        if (num_iteration == 2)
```

```
			return deltaDT;
		else if (num_iteration == 3){
			if(previous_output_flag){
			previous_departchoice_file.Rewind();
			while (previous_departchoice_file.Read ()) {
				Show_Progress ();
				p_person = previous_departchoice_file.Person ();
				p_hhold  = previous_departchoice_file.Hhold ();

				if (hhold == p_hhold && p_person==person && activitynumber ==
previous_departchoice_file.Activity()){
					p_DT = previous_departchoice_file.Dt();
					p_record_find=true;
					break;
				}
			}
			if (!p_record_find){
				Print (2, "DeltaDT:Household ID %d, person ID %d, and acitivity ID
%d was not found in the pervious_output file", hhold, person,activitynumber);
				return deltaDT;
			}
			}
			if (num_iteration>1 && !previous_output_flag)
				Error ("DeltaDT:This is the number of iteration %d, which is
greater than 1, you should specify one pervious output file ", num_iteration);

			deltaDT = p_DT;
		}
		else if(previous_output_flag && previous_previous_output_flag &&
num_iteration>2){

			previous_departchoice_file.Rewind();
			while (previous_departchoice_file.Read ()) {
				Show_Progress ();
				p_person = previous_departchoice_file.Person ();
				p_hhold  = previous_departchoice_file.Hhold ();

				//int tmp = previous_departchoice_file.Activity();

				if (hhold == p_hhold && p_person == person && activitynumber ==
previous_departchoice_file.Activity()){
					p_DT = previous_departchoice_file.Dt();
					p_record_find=true;
					break;
				}
			}
			p_previous_departchoice_file.Rewind();
```

```
              while (p_previous_departchoice_file.Read ()) {
                     Show_Progress ();
                     p_p_person = p_previous_departchoice_file.Person ();
                     p_p_hhold  = p_previous_departchoice_file.Hhold ();

                     if (hhold == p_p_hhold && p_p_person == person && activitynumber
== p_previous_departchoice_file.Activity()){
                            p_p_DT=p_previous_departchoice_file.Dt();
                         p_p_record_find = true;
                            break;
                     }
              }
              if (!p_p_record_find ){
                            Print (2,"DeltaDT:Household ID %d, person ID %d,and
acitivity ID %d was not found in the previous_pervious_output file", hhold,
person,activitynumber);
                      return deltaDT;
              }
                     if(!p_record_find)
                            Error ("DeltaDT:Household ID %d, person ID %d, and
acitivity ID %d was not found in the pervious_output file", hhold,
person,activitynumber);

                  deltaDT = p_DT-p_p_DT;
        }

           if (num_iteration>2 && !previous_output_flag )
                     Error ("DeltaDT:This is the number of iteration %d, which is
greater than 2, you should specify one pervious output file ", num_iteration);

              if (num_iteration>2 && !previous_previous_output_flag )
                     Error ("DeltaDT:This is the number of iteration %d, which is
greater than 2, you should specify one pervious_pervious output file ", num_iteration);

              if(deltaDT == 0)  //--- to ensure the denometor not zero ---
                     deltaDT = 1;

              return deltaDT;

}

//-----------------------------------------------------------
//     DeltaDT_FAST
//-----------------------------------------------------------
double DepartSelect::DeltaDT_FAST (int num_iteration, bool gender, int hhold, int
person, int activitynumber)
{
        double deltaDT = 1; double p_DT = 0, p_p_DT = 0;
    Departchoice_Data* Depart_ptr = NULL, *p_Depart_ptr = NULL, *p_p_Depart_ptr = NULL;

        if (num_iteration>1 && !previous_output_flag)
                Error ("DeltaTR:This is the number of iteration %d, which is greater than
1, you should specify one pervious output file ", num_iteration);

        if (num_iteration>2 && !previous_previous_output_flag )
                Error ("DeltaDT:This is the number of iteration %d, which is greater than
2, you should specify one pervious_pervious output file ", num_iteration);

        if (num_iteration == 1)
                return deltaDT;
        else if(previous_output_flag && num_iteration == 2){
                        Depart_ptr = pre_departchoice_data.Get(hhold*Traveler_Scale () +
person,activitynumber);
                        if(Depart_ptr == NULL){
                                Print (2, "this is not a valid record, we can not solve the
DeltaTR for hhold %d, person %d, activity number %d" ,hhold, person, activitynumber);
                                return deltaDT;
                        }

                        if(Depart_ptr->Is_Changed()){
                                p_DT = Depart_ptr->Dt();
                                deltaDT = p_DT;
                        }
                        else
                                return deltaDT;
                }
        else if(previous_output_flag && previous_previous_output_flag &&
num_iteration>2){
                        p_Depart_ptr = pre_departchoice_data.Get(hhold*Traveler_Scale () +
person,activitynumber);
                        if(p_Depart_ptr == NULL){
                                Print (2, "this is not a valid record for pre-dtc file, we
can not solve the DeltaDT for hhold %d, person %d, activity number %d" ,hhold, person,
activitynumber);
                                return deltaDT;
                        }
                        p_DT = p_Depart_ptr->Dt();

                        p_p_Depart_ptr =
pre_pre_departchoice_data.Get(hhold*Traveler_Scale () + person,activitynumber);
                        if(p_p_Depart_ptr == NULL){
                                Print (2, "this is not a valid record for pre-pre dtc file,
we can not solve the DeltaDT for hhold %d, person %d, activity number %d" ,hhold,
person, activitynumber);
```

```
				return deltaDT;
			}
			p_p_DT = p_p_Depart_ptr->Dt();

			if(p_Depart_ptr->Is_Changed() &&  p_p_Depart_ptr ->Is_Changed()){

				deltaDT = p_DT-p_p_DT;
			}
			else if((p_Depart_ptr->Is_Changed() == 1) &&  (p_p_Depart_ptr -
>Is_Changed() == 0)){
				deltaDT = p_DT;
			}
			else if((p_Depart_ptr->Is_Changed() == 0) &&  (p_p_Depart_ptr -
>Is_Changed() == 1)){
				deltaDT = p_p_DT;
			}
			else if((p_Depart_ptr->Is_Changed() == 0) &&  (p_p_Depart_ptr -
>Is_Changed() == 0)){
				return deltaDT;
			}
	}

	if(deltaDT == 0)  //--- to ensure the denometor not zero ---
			deltaDT = 1;

	return deltaDT;

}

//-----------------------------------------------------------
//	Cal_LMD
//-----------------------------------------------------------
double DepartSelect::Cal_LMD (int num_iteration, bool gender, int hhold, int person,
int activitynumber)
{
	double LMD = 1, p_DT = 0, p_p_DT = 0, deltaDT = 0; int p_hhold, p_person,
p_p_hhold, p_p_person; bool p_record_find = false, p_p_record_find = false;
	if (num_iteration == 1 || num_iteration == 2)
		LMD = 1;
	else{
		if(previous_output_flag){
		previous_departchoice_file.Rewind();
		while (previous_departchoice_file.Read ()) {
			Show_Progress ();
			p_person = previous_departchoice_file.Person ();
			p_hhold  = previous_departchoice_file.Hhold ();
```

```
                        if (hhold == p_hhold && p_person == person && activitynumber ==
previous_departchoice_file.Activity() ){
                                p_DT = previous_departchoice_file.Dt();
                                p_record_find=true;
                                break;
                        }
                }
                if (!p_record_find)
                        Error ("LMD:Household ID %d, person ID %d, and acitivity ID %d was
not found in the pervious_output file", hhold, person,activitynumber);
                }
                if (!previous_output_flag)
                        Error ("LMD:This is the number of iteration %d, which is greater
than 1, you should specify one pervious output file ", num_iteration);

                if(previous_previous_output_flag){
                p_previous_departchoice_file.Rewind();
                while (p_previous_departchoice_file.Read ()) {
                        Show_Progress ();
                        p_p_person = p_previous_departchoice_file.Person ();
                        p_p_hhold  = p_previous_departchoice_file.Hhold ();

                        if (hhold==p_p_hhold && p_p_person==person && activitynumber ==
p_previous_departchoice_file.Activity()){
                                p_p_DT=p_previous_departchoice_file.Dt();
                                p_p_record_find = true;
                                break;
                        }
                }
                        if (!p_p_record_find)
                                Error ("LMD:Household ID %d, person ID %d,and acitivity ID
%d was not found in the previous_pervious_output file", hhold, person,activitynumber);
                }
                        if (!previous_previous_output_flag)
                                Error ("LMD:this is the number of iteration %d, which is
greater than 2, you should specify one pervious_pervious output file ", num_iteration);

                        deltaDT = p_DT-p_p_DT;

                        if (deltaDT ==0)
                                LMD = 0;
                        else
                                LMD = 1;

                }
        return LMD;
}
```

```
//----------------------------------------------------------
//     Cal_LMD_FAST
//----------------------------------------------------------
double DepartSelect::Cal_LMD_FAST (int num_iteration, bool gender, int hhold, int
person, int activitynumber)
{
       double LMD = 0, p_DT = 0, p_p_DT = 0, deltaDT = 0;

    Departchoice_Data* Depart_ptr = NULL, *p_Depart_ptr = NULL, *p_p_Depart_ptr = NULL;

       if (num_iteration>2 && !previous_previous_output_flag )
                    Error ("LMD:This is the number of iteration %d, which is greater
than 2, you should specify one pervious_pervious output file ", num_iteration);

       if (num_iteration>2 && !previous_output_flag )
                    Error ("LMD:This is the number of iteration %d, which is greater
than 2, you should specify one pervious output file ", num_iteration);

       if (num_iteration == 1 || num_iteration == 2)
              return LMD;

    else if(previous_output_flag && previous_previous_output_flag && num_iteration>2){
                    p_Depart_ptr = pre_departchoice_data.Get(hhold*Traveler_Scale () +
person,activitynumber);
                    if(p_Depart_ptr == NULL){
                           Print (2, "this is not a valid record for pre-dtc file, we
can not solve the DeltaDT for hhold %d, person %d, activity number %d" ,hhold, person,
activitynumber);
                           return LMD;
                    }
                    p_DT = p_Depart_ptr->Dt();

                    p_p_Depart_ptr =
pre_pre_departchoice_data.Get(hhold*Traveler_Scale () + person,activitynumber);
                    if(p_p_Depart_ptr == NULL){
                           Print (2, "this is not a valid record for pre-pre dtc file,
we can not solve the DeltaDT for hhold %d, person %d, activity number %d" ,hhold,
person, activitynumber);
                           return LMD;
                    }
                    p_p_DT = p_p_Depart_ptr->Dt();

                    if(p_Depart_ptr->Is_Changed() == 0 &&  p_p_Depart_ptr -
>Is_Changed() == 0){
                           deltaDT = 0;
                    }
```

```
                else
                        return 1;
        }

        if (deltaDT == 0)
                LMD = 0;

        return LMD;
}

//---------------------------------------------------------
//      RandomNumGen
//---------------------------------------------------------
int DepartSelect::Binary_number_gen ()
{
        //random.Seed (200);
        //return random.Probability ();

        return random.Nonuniform_binary_number_generator (50, 1, 0);

        //return random.Nonuniform_binary_number_generator_Lee (50, 1, 0);

        //return random.Binary_number_generator ();
}

//---------------------------------------------------------
//      RandomNumGen
//---------------------------------------------------------
int DepartSelect::Nonuniform_binary_number_generator ()
{
        //random.Seed (200);
        //return random.Probability ();
        return random.Nonuniform_binary_number_generator (80, 10, 0);
}

//---------------------------------------------------------
//      Get_totalpopulation_in_activity
//---------------------------------------------------------
int DepartSelect::Get_totalpopulation_in_activity ()
{
        int hhold, person, i = 0, totalnumber_pop = 1;
        traveler = new char * [activity_data.Num_Records()];
        Activity_Data* activity_ptr = NULL;
    activity_ptr = activity_data.First();
    hhold = activity_ptr->Household();
    person = activity_ptr->Person();
    traveler[0]=new char[128]; //here we assume, the string "hholdperson" will not
```

```
exceed 128 characters
        sprintf(traveler[0], "%d%d", hhold,person);
        while((activity_ptr=activity_data.Next())!=NULL){
                i++;
                hhold = activity_ptr->Household();
                person = activity_ptr->Person();
                traveler[i]=new char[128];
                sprintf(traveler[i], "%d%d", hhold,person);
        }
     activity_ptr=activity_data.Last();
     hhold = activity_ptr->Household();
        person = activity_ptr->Person();
        i++;
        traveler[i]=new char[128];
        sprintf(traveler[i], "%d%d", hhold,person);

        for ( int j=0; j<i; j++){
                if (strcmp(traveler[j],traveler[j+1]))
                        totalnumber_pop++;
                else
                        continue;
        }
        clear_mem (activity_data.Num_Records());
        return totalnumber_pop;

}

//-----------------------------------------------------------
//      clear_mem
//-----------------------------------------------------------
void DepartSelect::clear_mem (int num_records )
{
        if (traveler != NULL) {
                        for (int i=0; i <= num_records; i++) {
                                if (traveler [i] != NULL) {
                                        delete [] traveler [i];
                                }
                        }
                        //delete [] traveler; //--- there is memory exception when trying
to delete the memory allocated to this
        }

}

//-----------------------------------------------------------
//      Cal_deltaT
//-----------------------------------------------------------
```

```
double DepartSelect::Cal_deltaT (int SD, double IBD)
{
	double deltaT=0;

	if (IBD!=0){
		if (SD>0)
			deltaT = -(300*SD/IBD);
		else
			deltaT = 300*abs(SD)/IBD;
	}
	else
		deltaT=300*SD; //--- need to disscuss with LEE

	return deltaT;
}

//----------------------------------------------------------
//	is_eligibletraveler_C_1
//----------------------------------------------------------
bool DepartSelect::Is_eligibletraveler_C_1 (int SD, double IBD)
{
	if(abs(SD)>abs(IBD))
		return 1;
	else
		return 0;
}

//----------------------------------------------------------
//	is_eligibletraveler_C_2
//----------------------------------------------------------
bool DepartSelect::Is_eligibletraveler_C_2 (int hhold, int person, int activitynumber)
{

	int mode = 0, eligiblemode = 2, eligiblepassengers = 0 ;
	bool eligible = false;
	//activitypurpose = Get_activitivityinfor(startinglocation, endinglocation, 1);
	Activity_Data* activity_ptr = NULL;
    activity_ptr = activity_data.First();
	if (activity_ptr->Household() == hhold && activity_ptr->Person() == person &&
activity_ptr->Activity() == activitynumber){
		if(activity_ptr->Mode() == eligiblemode && activity_ptr->Passengers() ==
eligiblepassengers)
			return true;
		else
			return false;
	}
	while((activity_ptr = activity_data.Next())!=NULL){
			if (activity_ptr->Household() == hhold && activity_ptr->Person() ==
person && activity_ptr->Activity() == activitynumber){
				if(activity_ptr->Mode() == eligiblemode && activity_ptr-
>Passengers() == eligiblepassengers){
						eligible = true;
						break;

				}
			}
		}
	if (eligible)
			return true;
		activity_ptr=activity_data.Last();
	 if (activity_ptr->Household() == hhold && activity_ptr->Person() == person &&
activity_ptr->Activity() == activitynumber){
			if(activity_ptr->Mode() == eligiblemode && activity_ptr->Passengers() ==
eligiblepassengers)
				eligible = true;
			else
				eligible = false;
		}
	 if(eligible)
			return true;
		else
			return false;
}

//---------------------------------------------------------
//	Is_eligibletraveler_C_2_FAST
//---------------------------------------------------------
bool DepartSelect::Is_eligibletraveler_C_2_FAST (int hhold, int person, int
activitynumber)
{
	Activity_Data* activity_ptr = NULL;
	int eligiblemode = 2, eligiblepassengers = 0 ;
	bool eligible = false;

	activity_ptr = activity_data.Get(hhold*Traveler_Scale () +
person,activitynumber);

	if(activity_ptr == NULL){
		Print (2, "this is not a valid record, we can not solve the eligibility2,
for hhold %d, person %d, activity number %d" ,hhold, person, activitynumber);
	    return false;
	}

	if(activity_ptr->Mode() == eligiblemode && activity_ptr->Passengers() ==
eligiblepassengers)
		eligible = true;
	else
		eligible = false;

	 return eligible;
}

//--------------------------------------------------------
//	is_eligibletraveler_C_3
//--------------------------------------------------------
bool DepartSelect::Is_eligibletraveler_C_3 (int hhold, int person, int activitynumber,
int deltaT)
{
    int starttime, endtime;
	bool eligible = false, delayed = false;
	 if(deltaT <0)
		delayed = true;
	else
		delayed = false;
	Activity_Data* activity_ptr = NULL, *pre_activity_ptr = NULL;
    pre_activity_ptr = activity_data.First();

	while((activity_ptr=activity_data.Next())!=NULL){
		if (activity_ptr->Household() == hhold && activity_ptr->Person() ==
person && activity_ptr->Activity() == activitynumber){

			if(pre_activity_ptr->Household() != hhold || pre_activity_ptr-
>Person() != person)
				Error ("We requrie each hhold and person has even number of
activities");

			if(delayed && pre_activity_ptr->Duration()> abs(deltaT)){
			eligible = true;
			break;
			}
			else if(!delayed){
           endtime = pre_activity_ptr->End_Time();
		starttime = activity_ptr->Start_Time();
			if ((starttime-endtime)<0)
				Print (2, "The next activity starting time is less than the
current activity ending time for hhold %d, person %d, activity number %d" ,hhold,
person, activitynumber);
			if ((starttime-endtime)> abs(deltaT)) {
				eligible = true;
				break;
			}
```

```
                }
            }
        pre_activity_ptr = activity_ptr;
         }
      activity_ptr=activity_data.Last();
         if (activity_ptr->Household() == hhold && activity_ptr->Person() == person &&
activity_ptr->Activity() == activitynumber){

                        if(pre_activity_ptr->Household() != hhold || pre_activity_ptr-
>Person() != person)
                                Error ("We requrie each hhold and person has even number of
activities");

                        if(delayed && pre_activity_ptr->Duration()> abs(deltaT)){
                        eligible = true;
                        }
                        else if(!delayed){
              endtime = pre_activity_ptr->End_Time();
                 starttime = activity_ptr->Start_Time();
                        if ((starttime-endtime)<0)
                                Print (2, "The next activity starting time is less than the
current activity ending time for hhold %d, person %d, activity number %d" ,hhold,
person, activitynumber);
                        if ((starttime-endtime)> abs(deltaT)) {
                                eligible = true;
                        }
                }
            }

        if (eligible)
                return true;
        else
                return false;

}

//------------------------------------------------------------
//     is_eligibletraveler_C_3
//------------------------------------------------------------
bool DepartSelect::Is_eligibletraveler_C_3_FAST (int hhold, int person, int
activitynumber, int deltaT)
{

        Activity_Data* activity_ptr = NULL, *pre_activity_ptr = NULL;
        int endtime, starttime, index ;
        bool eligible = false, delayed = false;
        if(deltaT <0)
                delayed = true;
        else
                delayed = false;

        activity_ptr = activity_data.Get(hhold*Traveler_Scale () +
person,activitynumber);

        if(activity_ptr == NULL){
                Print (2, "this is not a valid record, we can not solve the eligbility3,
for hhold %d, person %d, activity number %d" ,hhold, person, activitynumber);
            return false;
        }
        index = activity_data.Record_Index ();
     index = index-1;
        if(activity_data.Record_Index (index)){
                 pre_activity_ptr = activity_data.Record(index);
                 if (pre_activity_ptr->Household() == hhold && pre_activity_ptr->Person()
== person){
                        if(delayed && pre_activity_ptr->Duration()> abs(deltaT)){
                        eligible = true;
                        }
                        else if(!delayed){
             endtime = pre_activity_ptr->End_Time();
                starttime = activity_ptr->Start_Time();
                        if ((starttime-endtime)<0)
                                Print (2, "Elgible3:The next activity starting time is less
than the current activity ending time for hhold %d, person %d, activity number
%d" ,hhold, person, activitynumber);
                        if ((starttime-endtime)> abs(deltaT)) {
                                eligible = true;
                        }
                 }
          }
          else{
                  eligible = false;
                  Print (2, "Eligible3 Faliure: we can not find the previous record for
hhold %d, person %d, activity number %d" ,hhold, person, activitynumber);
         }
        }
        else{
         eligible = false;
                Print (2, "This is the first record, we can not find the good time space,
for hhold %d, person %d, activity number %d" ,hhold, person, activitynumber);
        }
        return eligible;
}
```

```
bool DepartSelect::Is_eligible (int SD, int IBD, int hhold, int person, int 
activitynumber)
{
	bool C1 = false, C2 = false, C3 = false;
	double deltaT = 0;
	C1 = Is_eligibletraveler_C_1 (SD, IBD);
	C2 = Is_eligibletraveler_C_2_FAST (hhold, person, activitynumber);
	deltaT = Cal_deltaT (SD, IBD);
	C3 = Is_eligibletraveler_C_3_FAST (hhold, person, activitynumber, deltaT);
	if(C1 && C2 && C3)
		return true;
	else
		return false;

}

//--------------------------------------------------------
//     Is_stop
//--------------------------------------------------------
bool DepartSelect::Is_stop (int SD, int IBD, int hhold, int person, int activitynumber)
{
	//int deltaT, activitypurpose;
	//activitypurpose = Get_activitivitypurpose (activitynumber);
	if(Is_eligible (SD, IBD, hhold, person, activitynumber))
		total_eligible_people++;
	if 
(100*total_eligible_people/Get_totalpopulation_in_activity()>=departchoice_percentage)
	 return true;
	else
	    return false;
}

//--------------------------------------------------------------------------------
//     Get_activitivityinfor, 0 activity number, 1 activity purpose
//  NOTE: currently, the startinglocation and endinglocation were used to seek the 
mached tactivity for a certain plan
//--------------------------------------------------------------------------------
int DepartSelect::Get_activitivityinfor (int startinglocation, int endinglocation, int 
returnflag, int hhold, int person)
{
    int location = 0,pre_activity_location = 0, activitynumber = 0, activitypurpose = 0, 
returnedvalue = 0, numberofactivities = 0;
	bool found = false;
	Activity_Data* activity_ptr = NULL, *pre_activityptr = NULL;
    activity_ptr = activity_data.First();
	pre_activityptr = activity_ptr;
```

```
        if (activity_ptr->Household() == hhold && activity_ptr->Person() == person &&
activity_ptr->Location() == endinglocation){
                Error ("The firstlocation for hhold %d, person %d should not be the
ending location ", hhold, person);
        }
   numberofactivities =0;
        while((activity_ptr = activity_data.Next())!=NULL){
                if(activity_ptr->Household() == hhold && activity_ptr->Person() == person
&& endinglocation == activity_ptr->Location()){
                        if(pre_activityptr->Household() == hhold && pre_activityptr-
>Person() == person && startinglocation == pre_activityptr->Location()){
                        activitynumber = activity_ptr->Activity();
                        activitypurpose = activity_ptr->Purpose();
                        found = true;
                        break;
                        }
                }
                pre_activityptr = activity_ptr;
        }
    activity_ptr=activity_data.Last();
        location = activity_ptr->Location();
        if(activity_ptr->Household() == hhold && activity_ptr->Person() == person &&
endinglocation == activity_ptr->Location()){
                   if(pre_activityptr->Household() == hhold && pre_activityptr->Person()
== person && startinglocation == pre_activityptr->Location()){
                        activitynumber = activity_ptr->Activity();
                        activitypurpose = activity_ptr->Purpose();
                        found = true;
                        }
        }

        if(!found)
                Error ("No activity matches the plan file plan");

        switch(returnflag){
                case  0:
                        returnedvalue = activitynumber;
                        break;
                case  1:
                        returnedvalue = activitypurpose;
                        break;
                default:
                        break;
        }

        return returnedvalue;
}
```

```
//-----------------------------------------------------------------------------------
//      Get_activitivityinfor_FAST, 0 activity number, 1 activity purpose
//  NOTE: currently, the startinglocation and endinglocation were used to seek the
mached tactivity for a certain plan
//-----------------------------------------------------------------------------------
int DepartSelect::Get_activitivityinfor_FAST (int startinglocation, int endinglocation,
int returnflag, int hhold, int person)
{
	int index, location = 0,pre_activity_location = 0, activitynumber = 0,
activitypurpose = 0, returnedvalue = 0, numberofactivities = 0;
	bool found = false, foundprerecord = false;
	int activitynumberarray[] ={0, 1, 2, 3, 4, 5, 6, 7,8,9,10,11,12,13, 14, 15,
16,17,18,19,20,21,22,23,24,25,26,27,28,29,30}; //--- assume, each person will not have
over 30 activities for a simulation trial

	Activity_Data* activity_ptr = NULL;
	for ( int i = 0; i <= 20; i++){
		activity_ptr = activity_data.Get(hhold*Traveler_Scale () +
person,activitynumberarray[i]);
		if(activity_ptr != NULL){
			if (activity_ptr->Household() == hhold && activity_ptr->Person()
== person && endinglocation == activity_ptr->Location() ){
				if(!Get_activitivityinfor_Child (activity_ptr,hhold,person,
startinglocation, activitynumber,activitypurpose))
					continue;
				else{
				    found = true;
					break;
			}
		}
	 }
	}
	if(!found){
		Error ("GetActivityInfor: No activity matches the plan file plan for
hhold %, person %d", hhold, person);
	}

	switch(returnflag){
		case  0:
			returnedvalue = activitynumber;
			break;
		case  1:
			returnedvalue = activitypurpose;
			break;
		default:
			break;
```

```
        }

        return returnedvalue;

}

//------------------------------------------------------------
//      Get_activitivityinfor_Child
//------------------------------------------------------------
bool DepartSelect::Get_activitivityinfor_Child (Activity_Data* pre_activity_ptr, int
hhold, int person, int startinglocation, int& activitynumber, int& activitypurpose)
{
        bool foundprerecord = false; int index;
        Activity_Data* activity_ptr = NULL;
        index = activity_data.Record_Index ();
     index = index-1;
        if(activity_data.Record_Index (index)){
                activity_ptr = activity_data.Record(index);
                if (activity_ptr->Household() == hhold && activity_ptr->Person() ==
person && startinglocation == activity_ptr->Location() ){
                        activitynumber = pre_activity_ptr->Activity();
                        activitypurpose = pre_activity_ptr->Purpose();
                        foundprerecord = true;
                }
                else
                        foundprerecord = false;
        }
        return foundprerecord;
}

//------------------------------------------------------------
//      Get_activitivitypurpose
//------------------------------------------------------------
int DepartSelect::Get_activitivitypurpose (int activitynumber, int hhold, int person)
{
     int activitypurpose = 0;
        bool found = false;
        Activity_Data* activity_ptr = NULL;
    activity_ptr = activity_data.First();
        if (activity_ptr->Household() == hhold && activity_ptr->Person() == person &&
activity_ptr->Activity() == activitynumber)
               return activity_ptr->Purpose();
        while((activity_ptr = activity_data.Next())!=NULL){
               if(activity_ptr->Household() == hhold && activity_ptr->Person() == person
&& activity_ptr->Activity() == activitynumber){
                        activitypurpose = activity_ptr->Purpose();
                        found = true;
                        break;
                }
        }
    activity_ptr=activity_data.Last();
        if(activity_ptr->Household() == hhold && activity_ptr->Person() == person &&
activity_ptr->Activity() == activitynumber){
                        activitypurpose = activity_ptr->Purpose();
                        found = true;
        }

        if(!found)
                Error ("No activity matches the plan file plan");

        return activitypurpose;
}

//-----------------------------------------------------------
//      Get_activitivitypurpose_FAST
//-----------------------------------------------------------
int DepartSelect::Get_activitivitypurpose_FAST (int activitynumber, int hhold, int
person)
{
    int activitypurpose = 0;
        Activity_Data* activity_ptr = NULL;

        activity_ptr = activity_data.Get(hhold*Traveler_Scale () +
person,activitynumber);

        if(activity_ptr == NULL){
                Print (2,"did not sucessfully activity purpose for hhold %d, person %d,
and activity %d", hhold, person, activitynumber);
                return activitypurpose;
        }
         activitypurpose = activity_ptr->Purpose();

         return activitypurpose;
}

//-----------------------------------------------------------
//      is_specifiedactivitypurpose
//-----------------------------------------------------------
bool DepartSelect::is_specifiedactivitypurpose (int activitypurpose)
{
        int constant_activitypurpose[] = {1, 7, 8, 9, 11, 17, 18, 19};

        bool found = false;
        for (int i=0; i<=7; i++){
                if(activitypurpose != constant_activitypurpose[i])
                        continue;
                else {
                        found = true;
                        break;
                }
        }
        return found;
}

//------------------------------------------------------------
//      Get_activity_ptr,depleted
//------------------------------------------------------------
Activity_Data* DepartSelect::Get_activity_ptr (int hhold, int person, int 
activitynumber)
{

        bool find = false;
        //activitypurpose = Get_activitivityinfor(startinglocation, endinglocation, 1);
        Activity_Data* activity_ptr = NULL, *find_activity_ptr = NULL;
    activity_ptr = activity_data.First();
        if (activity_ptr->Household() == hhold && activity_ptr->Person() == person && 
activity_ptr->Activity() == activitynumber){
                find_activity_ptr = activity_ptr;
                find = true;
        }

        while((activity_ptr = activity_data.Next())!=NULL){
                if (activity_ptr->Household() == hhold && activity_ptr->Person() == 
person && activity_ptr->Activity() == activitynumber){
                                find_activity_ptr = activity_ptr;
                                   find = true;
                                      break;

                        }
                }
  if (find)
                return find_activity_ptr;
        activity_ptr=activity_data.Last();
        if (activity_ptr->Household() == hhold && activity_ptr->Person() == person && 
activity_ptr->Activity() == activitynumber){
                find_activity_ptr = activity_ptr;
                find = true;
        }
        if(!find)
                Error ("No activity matches the activity file, so no activity record can
be possibly adjusted for HHOLD %d, PERSON %d, and Activity %d", hhold, person,
activitynumber);
       else
              return find_activity_ptr;
}

//---------------------------------------------------------
//     Setup_pre_depart_array
//---------------------------------------------------------
void DepartSelect::Setup_pre_depart_array (void)
{
       Departchoice_Data* depart_ptr = NULL, *depart_ptr_tmp = NULL;

       depart_ptr = pre_departchoice_data.New_Record (true);
       depart_ptr_tmp = departchoice_data.First();
       copy_depart_ptr (depart_ptr, depart_ptr_tmp);
       pre_departchoice_data.Add ();

       depart_ptr = pre_departchoice_data.New_Record (true);
       while((depart_ptr_tmp=departchoice_data.Next())!=NULL){
              copy_depart_ptr (depart_ptr, depart_ptr_tmp);
              pre_departchoice_data.Add ();
              depart_ptr = pre_departchoice_data.New_Record (true);
       }

       depart_ptr = pre_departchoice_data.New_Record (true);
       depart_ptr_tmp=departchoice_data.Last();
    copy_depart_ptr (depart_ptr, depart_ptr_tmp);
       pre_departchoice_data.Add ();

}

//---------------------------------------------------------
//     Setup_pre_pre_depart_array
//---------------------------------------------------------
void DepartSelect::Setup_pre_pre_depart_array (void)
{

       Departchoice_Data* depart_ptr = NULL, *depart_ptr_tmp = NULL;

       depart_ptr = pre_pre_departchoice_data.New_Record (true);
       depart_ptr_tmp = departchoice_data.First();
       copy_depart_ptr (depart_ptr, depart_ptr_tmp);
       pre_pre_departchoice_data.Add ();

       depart_ptr = pre_pre_departchoice_data.New_Record (true);
       while((depart_ptr_tmp=departchoice_data.Next())!=NULL){
```

```
                copy_depart_ptr (depart_ptr, depart_ptr_tmp);
                pre_pre_departchoice_data.Add ();
                depart_ptr = pre_pre_departchoice_data.New_Record (true);
        }

        depart_ptr = pre_pre_departchoice_data.New_Record (true);
        depart_ptr_tmp=departchoice_data.Last();
    copy_depart_ptr (depart_ptr, depart_ptr_tmp);
        pre_pre_departchoice_data.Add ();
}


//-----------------------------------------------------------
//      Setup_new_activity_array
//-----------------------------------------------------------
void DepartSelect::Setup_new_activity_array (void)
{
        Activity_Data* activity_ptr = NULL, *activity_ptr_tmp = NULL;

        new_activity_array = new Activity_Array;

        if (new_activity_array ==NULL)
                        Error ("Not able to allocate memory for new activity records");

        activity_ptr = new_activity_data.New_Record (true);

        activity_ptr_tmp = activity_data.First();
        copy_activity_ptr (activity_ptr, activity_ptr_tmp);
        new_activity_data.Add ();

        activity_ptr = new_activity_data.New_Record (true);
        while((activity_ptr_tmp=activity_data.Next())!=NULL){
                copy_activity_ptr (activity_ptr, activity_ptr_tmp);
                new_activity_data.Add ();
                activity_ptr = new_activity_data.New_Record (true);
        }

        activity_ptr = new_activity_data.New_Record (true);
        activity_ptr_tmp=activity_data.Last();
    copy_activity_ptr (activity_ptr, activity_ptr_tmp);
        new_activity_data.Add ();

}

//-----------------------------------------------------------
//      copy_activity_ptr
//-----------------------------------------------------------
```

```
void DepartSelect::copy_activity_ptr (Activity_Data* activity_ptr, Activity_Data*
activity_ptr_tmp)
{
		activity_ptr->Traveler(activity_ptr_tmp->Traveler());
		activity_ptr->Household(activity_ptr_tmp->Household());
    activity_ptr->Person(activity_ptr_tmp->Person());
    activity_ptr->Activity(activity_ptr_tmp->Activity());
    activity_ptr->Purpose(activity_ptr_tmp->Purpose());
    activity_ptr->Priority(activity_ptr_tmp->Priority());
    activity_ptr->Start_Time(activity_ptr_tmp->Start_Time());
    activity_ptr->End_Time(activity_ptr_tmp->End_Time());
    activity_ptr->Duration(activity_ptr_tmp->Duration());
    activity_ptr->Mode(activity_ptr_tmp->Mode());
    activity_ptr->Vehicle(activity_ptr_tmp->Vehicle());
    activity_ptr->Location(activity_ptr_tmp->Location());
		activity_ptr->Passengers(activity_ptr_tmp->Passengers());
    activity_ptr->Constraint(activity_ptr_tmp->Constraint());
    activity_ptr->Problem(activity_ptr_tmp->Problem());
}

//---------------------------------------------------------
//	copy_depart_ptr
//---------------------------------------------------------
void DepartSelect::copy_depart_ptr (Departchoice_Data* depart_ptr, Departchoice_Data*
depart_ptr_tmp)
{
		depart_ptr->Hhold (depart_ptr_tmp->Hhold());
    depart_ptr->Person (depart_ptr_tmp->Person());
    depart_ptr->Age (depart_ptr_tmp->Age());
    depart_ptr->Gender (depart_ptr_tmp->Gender());
    depart_ptr->Activity (depart_ptr_tmp->Activity());
    depart_ptr->Et (depart_ptr_tmp->Et());
		depart_ptr->Rt (depart_ptr_tmp->Rt());
    depart_ptr->Sd (depart_ptr_tmp->Sd());
    depart_ptr->Dtr (depart_ptr_tmp->Dtr());
    depart_ptr->Ddt (depart_ptr_tmp->Ddt());
		depart_ptr->Ibd (depart_ptr_tmp->Ibd());
    depart_ptr->Utility (depart_ptr_tmp->Utility());
    depart_ptr->Dt (depart_ptr_tmp->Dt());
    depart_ptr->Ddt (depart_ptr_tmp->Ddt());
    depart_ptr->Flexibility (depart_ptr_tmp->Flexibility());
    depart_ptr->Retire (depart_ptr_tmp->Retire());
    depart_ptr->Flextimework (depart_ptr_tmp->Flextimework());
    depart_ptr->Income (depart_ptr_tmp->Income());
		depart_ptr->Trippurpose (depart_ptr_tmp->Trippurpose());
		depart_ptr->Is_Changed (depart_ptr_tmp->Is_Changed ());
```

```
}

//--------------------------------------------------------
//     Change_activity_ptr
//--------------------------------------------------------
void DepartSelect::Change_activity_ptr (int hhold, int person, int activitynumber, int
deltaT)
{
       bool eligible = false, delayed = false;    int Hour_Seconds = 1;

       if(activity_file_hour_flag)
              Hour_Seconds = 3600;

       if(deltaT <0)
              delayed = true;
       else
              delayed = false;

       Activity_Data* activity_ptr = NULL, *pre_activity_ptr =NULL;
    pre_activity_ptr = new_activity_data.First();

       while((activity_ptr=new_activity_data.Next())!=NULL){
              if (activity_ptr->Household() == hhold && activity_ptr->Person() ==
person && activity_ptr->Activity() == activitynumber){
                     if(    pre_activity_ptr == NULL)
                     Error ("NO pre-activity record found for ending time update,
within chang-activity-ptr function");
                     if(delayed){
                     pre_activity_ptr->End_Time( (pre_activity_ptr->End_Time() +
deltaT)/Hour_Seconds );
                     pre_activity_ptr->Duration( (pre_activity_ptr->Duration() +
deltaT)/Hour_Seconds);
                     break;
                     }
                     else if(!delayed){
                     pre_activity_ptr->End_Time((pre_activity_ptr->End_Time() +
deltaT)/Hour_Seconds);
                     pre_activity_ptr->Duration((pre_activity_ptr->End_Time() +
deltaT)/Hour_Seconds);
                     break;
                     }

          }
          pre_activity_ptr = activity_ptr;
       }

       activity_ptr=activity_data.Last();
```

```
        if (activity_ptr->Household() == hhold && activity_ptr->Person() == person &&
activity_ptr->Activity() == activitynumber){
                if(     pre_activity_ptr == NULL)
                        Error ("NO pre-activity record found for ending time update,
within chang-activity-ptr function");
                if(delayed){
                        pre_activity_ptr->End_Time( (pre_activity_ptr->End_Time() +
deltaT)/Hour_Seconds);
                        pre_activity_ptr->Duration( (pre_activity_ptr->End_Time() +
deltaT)/Hour_Seconds);
                        }
                else if(!delayed){
                        pre_activity_ptr->End_Time( (pre_activity_ptr->End_Time() +
deltaT)/Hour_Seconds);
                        pre_activity_ptr->Duration( (pre_activity_ptr->End_Time() +
deltaT)/Hour_Seconds);

                 }
         }

}

//-----------------------------------------------------------
//      Change_activity_ptr_FAST
//-----------------------------------------------------------
void DepartSelect::Change_activity_ptr_FAST (int hhold, int person, int activitynumber,
int deltaT)
{
        int activitypurpose = 0, index;
        bool delayed = false;

        if(deltaT <0)
                delayed = true;
        else
                delayed = false;

        Activity_Data* activity_ptr = NULL;
        activity_ptr = new_activity_data.Get(hhold*Traveler_Scale () +
person,activitynumber);

        if(activity_ptr == NULL){
                Print (2, "this is not a valid record, we can not update new activity
file, for hhold %d, person %d, activity number %d" ,hhold, person, activitynumber);
            return;
        }
        index = new_activity_data.Record_Index ();
    index = index-1;
```

```
        if(new_activity_data.Record_Index (index)){
                activity_ptr = new_activity_data.Record(index);
                if (activity_ptr->Household() == hhold && activity_ptr->Person() ==
person){
                        if(delayed){
                        activity_ptr->End_Time( (activity_ptr->End_Time() + deltaT));
                        activity_ptr->Duration( (activity_ptr->Duration() + deltaT));
                        //activity_ptr->Start_Time(activity_ptr->Start_Time());
                        }
                else if(!delayed){
                        activity_ptr->End_Time( (activity_ptr->End_Time() + deltaT));
                        activity_ptr->Duration( (activity_ptr->Duration() + deltaT));
                        //activity_ptr->Start_Time(activity_ptr->Start_Time());
                }
        }
        else{
            Print (2, "The new time space can not be applied to new activity file :pre-
record not existed  for hhold %d, person %d, activity number %d" ,hhold, person,
activitynumber);
         }
        }
        else{
                Print (2, "This is the first record, we can not update thivity file, for
hhold %d, person %d, activity number %d" ,hhold, person, activitynumber);
        }

}

//-----------------------------------------------------------
//      Get_Randomnumber
//-----------------------------------------------------------
void DepartSelect::Get_Randomnumber(int hhold, int person, int activitynumber, int&
flexibility, int& flextimework )
{
    bool found = false;
        Randomnumber_Data* random_ptr = NULL;
    random_ptr = randomnumber_data.First();
        if (random_ptr->Hhold() == hhold && random_ptr->Person() == person &&
random_ptr->Activity() == activitynumber){

                flexibility = random_ptr->Flexibility();
         flextimework = random_ptr->Flextimework();
                found =true;
        }
        while((random_ptr = randomnumber_data.Next())!=NULL){
                if (random_ptr->Hhold() == hhold && random_ptr->Person() == person &&
random_ptr->Activity() == activitynumber){
        flexibility = random_ptr->Flexibility();
         flextimework = random_ptr->Flextimework();
                found = true;
                break;
           }
        }
    random_ptr= randomnumber_data.Last();
        if (random_ptr->Hhold() == hhold && random_ptr->Person() == person &&
random_ptr->Activity() == activitynumber){
        flexibility = random_ptr->Flexibility();
         flextimework = random_ptr->Flextimework();
                found = true;
         }

        if(!found){
                flexibility = 1;  //--- if we did not find the suitable random number
from file, then we initilize one for them. ---
         flextimework = 1; //--- if we did not find the suitable random number from file,
then we initilize one for them. ---
                Print (2, "Did not find the appropriate random number for hhold %d,
person %d, and activity %d", hhold, person, activitynumber);
        }
}

//----------------------------------------------------------
//      Get_Randomnumber_FAST
//----------------------------------------------------------
void DepartSelect::Get_Randomnumber_FAST (int hhold, int person, int activitynumber,
int& flexibility, int& flextimework )
{
        Randomnumber_Data* random_ptr = NULL;

        random_ptr = randomnumber_data.Get(hhold*Traveler_Scale () +
person,activitynumber);

        if(random_ptr == NULL){
                Print (2, "did not sucessfully find the suitable random number for hhold
%d, person %d, and activity %d, and they were replaced with 1,1", hhold, person,
activitynumber);
         flexibility = 1;
         flextimework = 1;
                return ;
        }
        flexibility = random_ptr->Flexibility();
    flextimework = random_ptr->Flextimework();
}
```

```
//--------------------------------------------------------
//     Activity_time_transform_H_S, hour to second
//--------------------------------------------------------
void DepartSelect::Activity_time_transform_H_S()
{
    int Hour_Seconds = 3600;
       Activity_Data* activity_ptr = NULL, *orginal_activity_ptr = NULL;

       activity_ptr = new_activity_data.First();
       activity_ptr->Start_Time(activity_ptr->Start_Time()*Hour_Seconds);
    activity_ptr->End_Time(activity_ptr->End_Time()*Hour_Seconds);
       activity_ptr->Duration(activity_ptr->Duration()*Hour_Seconds);

       orginal_activity_ptr = activity_data.First();
       orginal_activity_ptr->Start_Time(orginal_activity_ptr-
>Start_Time()*Hour_Seconds);
    orginal_activity_ptr->End_Time(orginal_activity_ptr->End_Time()*Hour_Seconds);
       orginal_activity_ptr->Duration(orginal_activity_ptr->Duration()*Hour_Seconds);

       while((activity_ptr = new_activity_data.Next())!=NULL){
              activity_ptr->Start_Time(activity_ptr->Start_Time()*Hour_Seconds);
        activity_ptr->End_Time(activity_ptr->End_Time()*Hour_Seconds);
              activity_ptr->Duration(activity_ptr->Duration()*Hour_Seconds);

              orginal_activity_ptr = activity_data.Next();
              if(orginal_activity_ptr != NULL){
        orginal_activity_ptr->Start_Time(orginal_activity_ptr-
>Start_Time()*Hour_Seconds);
        orginal_activity_ptr->End_Time(orginal_activity_ptr->End_Time()*Hour_Seconds);
              orginal_activity_ptr->Duration(orginal_activity_ptr-
>Duration()*Hour_Seconds);
              }
       }
    activity_ptr= new_activity_data.Last();
       activity_ptr->Start_Time(activity_ptr->Start_Time()*Hour_Seconds);
    activity_ptr->End_Time(activity_ptr->End_Time()*Hour_Seconds);
       activity_ptr->Duration(activity_ptr->Duration()*Hour_Seconds);

       orginal_activity_ptr= activity_data.Last();
       orginal_activity_ptr->Start_Time(orginal_activity_ptr-
>Start_Time()*Hour_Seconds);
    orginal_activity_ptr->End_Time(orginal_activity_ptr->End_Time()*Hour_Seconds);
       orginal_activity_ptr->Duration(orginal_activity_ptr->Duration()*Hour_Seconds);
}

//--------------------------------------------------------
//     Activity_time_transform_H_S_FAST, hour to second
```

```
//------------------------------------------------------------
void DepartSelect::Activity_time_transform_H_S_FAST(int hhold, int person, int
activitynumber)
{
    bool found = false; int Hour_Seconds = 3600, index = 0;
       Activity_Data* activity_ptr = NULL;

       activity_ptr = activity_data.Get(hhold*Traveler_Scale () +
person,activitynumber);

       if(activity_ptr == NULL){
              Print (2, "did not sucessfully transformed activity time (from hours to
seconds)for hhold %d, person %d, and activity %d", hhold, person, activitynumber);
           return ;
       }
       else{
           activity_ptr->Start_Time(activity_ptr->Start_Time()*Hour_Seconds);
        activity_ptr->End_Time(activity_ptr->End_Time()*Hour_Seconds);
              activity_ptr->Duration(activity_ptr->Duration()*Hour_Seconds);
       }
}

//------------------------------------------------------------
//     Activity_time_transform_S_H, second to hour, depleted
//------------------------------------------------------------
void DepartSelect::Activity_time_transform_S_H ()
{
    int Hour_Seconds = 3600;
       Activity_Data* activity_ptr = NULL;
    activity_ptr = new_activity_data.First();
       activity_ptr->Start_Time(activity_ptr->Start_Time()/Hour_Seconds);
    activity_ptr->End_Time(activity_ptr->End_Time()/Hour_Seconds);
       activity_ptr->Duration(activity_ptr->Duration()/Hour_Seconds);

       while((activity_ptr = new_activity_data.Next())!=NULL){
              activity_ptr->Start_Time(activity_ptr->Start_Time()/Hour_Seconds);
        activity_ptr->End_Time(activity_ptr->End_Time()/Hour_Seconds);
              activity_ptr->Duration(activity_ptr->Duration()/Hour_Seconds);
       }
    activity_ptr= new_activity_data.Last();
       activity_ptr->Start_Time(activity_ptr->Start_Time()/Hour_Seconds);
    activity_ptr->End_Time(activity_ptr->End_Time()/Hour_Seconds);
       activity_ptr->Duration(activity_ptr->Duration()/Hour_Seconds);

}

//------------------------------------------------------------
```

```
//      Write_output_file
//----------------------------------------------------------
void DepartSelect::Write_output_file (FILE* current_dtc_output_file, Output_data*
datatmp)
{
        fprintf (current_dtc_output_file,
"%d₩t%d₩t%d₩t%d₩t%d₩t%d₩t%d₩t%d₩t%d₩t%d₩t%.2f₩t%.2f₩t%.2f₩t%.2f₩t%d₩t%d₩t%d₩t%d₩t%d₩n",
                datatmp->hhold,
                datatmp->person,datatmp->age, datatmp->gender, datatmp->income, datatmp-
>activity,
                datatmp->Et, datatmp->Rt,
                datatmp->Sd, datatmp->Dtr, datatmp->Ddt, datatmp->Ibd,
                datatmp->utility, datatmp->dt,
                datatmp->flexibility, datatmp->retire, datatmp->flextimework,
                datatmp->trippurpose,datatmp->is_changed);
}

//----------------------------------------------------------
//      Write_output_file_header
//----------------------------------------------------------
void DepartSelect::Write_output_file_header (FILE* current_dtc_output_file)
{
        fprintf (current_dtc_output_file,
"%s₩t%s₩t%s₩t%s₩t%s₩t%s₩t%s₩t%s₩t%s₩t%s₩t%s₩t%s₩t%s₩t%s₩t%s₩t%s₩t%s₩t%s₩t%s₩n",  "hhold",
"person","age", "gender", "income", "activity", "Et","Rt",
                "Sd", "Dtr", "Ddt", "Ibd", "utility", "dt", "flexibility", "retire",
"flextimework","trippurpose", "ischanged");
}

//----------------------------------------------------------
//      Write_output_new_activityfile
//----------------------------------------------------------
void DepartSelect::Write_output_new_activityfile (FILE* new_activity_file)
{
        Activity_Data* activity_ptr = NULL;

        activity_ptr = new_activity_data.First();
       fprintf (new_activity_file, "%d₩t%d₩t%d₩t%d₩t%.2f₩t%.2f₩t%.2f₩t%d₩t%d₩t%d₩t%d₩n",
activity_ptr->Household(), activity_ptr->Person(),activity_ptr->Activity()
                , activity_ptr->Purpose(), activity_ptr->Start_Time(),activity_ptr-
>End_Time(),
                activity_ptr->Duration(),activity_ptr->Mode(),activity_ptr-
>Vehicle(),activity_ptr->Location(),
                activity_ptr->Passengers());


        while((activity_ptr=new_activity_data.Next())!=NULL){
                fprintf (new_activity_file,
"%d₩t%d₩t%d₩t%d₩t%.2f₩t%.2f₩t%.2f₩t%d₩t%d₩t%d₩t%d₩n", activity_ptr->Household(),
activity_ptr->Person(),activity_ptr->Activity()
                , activity_ptr->Purpose(), activity_ptr->Start_Time(),activity_ptr-
>End_Time(),
                activity_ptr->Duration(),activity_ptr->Mode(),activity_ptr-
>Vehicle(),activity_ptr->Location(),
                activity_ptr->Passengers());
        }

        activity_ptr=new_activity_data.Last();
     fprintf (new_activity_file, "%d₩t%d₩t%d₩t%d₩t%.2f₩t%.2f₩t%.2f₩t%d₩t%d₩t%d₩t%d₩n",
activity_ptr->Household(), activity_ptr->Person(),activity_ptr->Activity()
        , activity_ptr->Purpose(), activity_ptr->Start_Time(),activity_ptr->End_Time(),
        activity_ptr->Duration(),activity_ptr->Mode(),activity_ptr-
>Vehicle(),activity_ptr->Location(),
        activity_ptr->Passengers());

}

//-----------------------------------------------------------
//      Write_output_new_activityfile_header
//-----------------------------------------------------------
void DepartSelect::Write_output_new_activityfile_header (FILE* new_activity_file)
{
        fprintf (new_activity_file, "%s₩t%s₩t%s₩t%s₩t%s₩t%s₩t%s₩t%s₩t%s₩t%s₩t%s₩n",
"HHOLD", "PERSON","ACTIVITY", "PURPOSE","START",
                "END", "DURATION", "MODE", "VEHICLE", "LOCATION", "PASSENGERS");
}

//-----------------------------------------------------------
//      Write_output_hhold_list
//-----------------------------------------------------------
void DepartSelect::Write_output_hhold_list (FILE* dtc_hhold_file, int num_sel,
Integer_Sort hhold_list)
{
        int num = 0, hh; bool select =false; double share;

        if (num_hh < 1) num_hh = 1;

        Print (1, "₩tNumber of Households Selected = %d (%.1lf%%)",
                num_sel, (100.0 * num_sel / num_hh));

        //---- select the households to re-route ----

        select = select_flag;
        share = percent;
        //---- check the maximum selection ----

        if (num_sel * percent / num_hh > max_percent) {
                select = true;
                share = max_percent * num_hh / num_sel;
        }

        //---- write the selected household ids ----

    num_hhold = 0;

    for (hh = hhold_list.First (); hh != 0; hh = hhold_list.Next ()) {
                if (select) {
                        if (random.Probability () > share) continue;
                }
                fprintf (dtc_hhold_file, "%d₩n", hh);
                num_hhold++;
                num++;
        }
        fclose (dtc_hhold_file);

        Print (1, "₩tNumber of Households Written = %d (%.1lf%%)",
                num, (100.0 * num / num_hh));

        //Write (1, "₩tNumber of Household IDs Written = %d", num_hhold);

}

//---------------------------------------------------------
//      Close_files
//---------------------------------------------------------
void DepartSelect::Close_files ()
{
     fclose (current_dtc_output_file);

     if (new_activity_file_flag)
          fclose (new_activity_file);

     fclose (before_dtc_hhold_file);

    if(use_of_after_dtc_holdlistfile_flag)
        fclose (after_dtc_hhold_file);

     plan_file.Close ();

}
```

< /Departselect/IBD_Cal.cpp >

```
//**********************************************************
//     IBD_Cal.cpp - calculate individual indifference band
//**********************************************************
#include "DepartSelect.hpp"
#include "In_Polygon.hpp"
#include "Utility.hpp"

//----------------------------------------------------------
//     IBD_Cal
//----------------------------------------------------------
#define constant_WI 12.258
#define constant_1_WI 6.71
#define constant_WI_AGEI 0.1663
#define constant_1_WI_AGEI 0.1023
#define constant_WI_GENDERI 1.774
#define constant_1_WI_GENDERI 1.5238
#define constant_WI_LMD_ETRATIO 0.425
#define constant_1_WI_LMD_RTRATIO 0.1435

//--- The units of IBD is min ---
double DepartSelect::IBD_Cal (int SD, int age, bool gender, int deltaTR, double deltaDT,
int LMD)
{
       int WI=0, ageconstant = 0; double IBD=0;
       if (SD>0)
              WI=1;
       if(age<20)
              ageconstant = 1;
       else if(20 <= age && age <= 39)
              ageconstant = 2;
       else if(40 <= age && age<= 59)
              ageconstant = 3;
       else if(60 <= age)
              ageconstant = 4;

       IBD=WI*constant_WI+constant_1_WI*(1-WI)+constant_WI_AGEI*WI*ageconstant+
              constant_1_WI_AGEI*(1-WI)*ageconstant-
              constant_WI_GENDERI*WI*gender-constant_1_WI_GENDERI*(1-WI)*gender+
              constant_WI_LMD_ETRATIO*WI*LMD*(deltaTR/deltaDT)+
              constant_1_WI_LMD_RTRATIO*(1-WI)*LMD*(deltaTR/deltaDT)+
random.Range_number_generator(-4,1);

       return IBD*60/2;  //--- transfer to seconds ---

}
```

< /Departselect/Readfiles.cpp >

```
//**********************************************************
//      Population.cpp - Read Population File
//**********************************************************

#include "DepartSelect.hpp"
#include "Utility.hpp"

//----------------------------------------------------------
//      Read_Population
//----------------------------------------------------------
void DepartSelect::Read_Population (void)
{
        //Db_File *file = Demand_Db_File (POPULATION);

    Population_Data *population_ptr;


        Show_Message ("Reading %s -- Record", pop_file.File_Type ());
        Set_Progress (10000);

        while (pop_file.Read ()) {
                Show_Progress ();

                population_ptr = population_data.New_Record (true);
                if (population_ptr == NULL) goto mem_error;
                population_ptr->Household (pop_file.Household ());
                population_ptr->Person (pop_file.Person ());
                population_ptr->Income ( pop_file.Income());
                population_ptr->Age (pop_file.Age());
                population_ptr->Gender ( pop_file.Gender());
         if (!population_data.Add ()) {
                                Error ("Adding Record to the Population Data Array");
                        }

                }
        End_Progress ();
        pop_file.Close();
        Print (2, "Number of Population Records Read = %d", Progress_Count ());
        return;

mem_error:
        Error ("Insufficient Memory for Population Data");
        return;
}
```

```
//---------------------------------------------------------
//      Read_pre_depart_file
//---------------------------------------------------------
void DepartSelect::Read_pre_depart_file (void)
{
    Departchoice_Data *Departchoice_ptr;

        if (departchoice_array == NULL) {
                        departchoice_array = new Departchoice_Array ();
                }
        if (pre_departchoice_array == NULL) {
                        pre_departchoice_array = new Departchoice_Array ();
        }

        Show_Message ("Reading %s -- Record", previous_departchoice_file.File_Type ());
        Set_Progress (10000);

        while (previous_departchoice_file.Read ()) {
                Show_Progress ();

                Departchoice_ptr = departchoice_data.New_Record (true);
                if (Departchoice_ptr == NULL) goto mem_error;
                Departchoice_ptr->Hhold(previous_departchoice_file.Hhold());
                Departchoice_ptr->Person(previous_departchoice_file.Person());
                Departchoice_ptr->Activity (previous_departchoice_file.Activity ());
                Departchoice_ptr->Et(previous_departchoice_file.Et());
                Departchoice_ptr->Rt(previous_departchoice_file.Rt());
                Departchoice_ptr->Sd (previous_departchoice_file.Sd ());
                Departchoice_ptr->Dtr(previous_departchoice_file.Dtr());
                Departchoice_ptr->Ddt(previous_departchoice_file.Ddt());
                Departchoice_ptr->Ibd (previous_departchoice_file.Ibd ());
                Departchoice_ptr->Utility(previous_departchoice_file.Utility());
                Departchoice_ptr->Dt (previous_departchoice_file.Dt ());

                Departchoice_ptr->Age (previous_departchoice_file.Age ());
                Departchoice_ptr->Gender (previous_departchoice_file.Gender ());
               Departchoice_ptr->Flexibility (previous_departchoice_file.Flexibility ());
                Departchoice_ptr->Retire (previous_departchoice_file.Retire ());
                Departchoice_ptr->Flextimework (previous_departchoice_file.Flextimework
());
                Departchoice_ptr->Income (previous_departchoice_file.Income ());
               Departchoice_ptr->Trippurpose (previous_departchoice_file.Trippurpose ());
                Departchoice_ptr->Is_Changed (previous_departchoice_file.Is_Changed ());


         if (!departchoice_data.Add ()) {
```

```
                              Error ("Adding Record to the pre-depart Data Array");
                       }

               }
        End_Progress ();
        //previous_departchoice_file.Close();
        Print (2, "Number of pre-depart Records Read = %d", Progress_Count ());
        return;

mem_error:
        Error ("Insufficient Memory for pre-depart Data");
        return;
}

//------------------------------------------------------------
//      Read_pre_pre_depart_file
//------------------------------------------------------------
void DepartSelect::Read_pre_pre_depart_file (void)
{
         Departchoice_Data *Departchoice_ptr;

        if (departchoice_array == NULL) {
                       departchoice_array = new Departchoice_Array ();
        }
        if (pre_pre_departchoice_array == NULL) {
                       pre_pre_departchoice_array = new Departchoice_Array ();
        }
       Show_Message ("Reading %s -- Record", p_previous_departchoice_file.File_Type ());
        Set_Progress (10000);

        while (p_previous_departchoice_file.Read ()) {
               Show_Progress ();

               Departchoice_ptr = departchoice_data.New_Record (true);
               if (Departchoice_ptr == NULL) goto mem_error;
               Departchoice_ptr->Hhold(p_previous_departchoice_file.Hhold());
               Departchoice_ptr->Person(p_previous_departchoice_file.Person());
               Departchoice_ptr->Activity (p_previous_departchoice_file.Activity ());
               Departchoice_ptr->Et(p_previous_departchoice_file.Et());
               Departchoice_ptr->Rt(p_previous_departchoice_file.Rt());
               Departchoice_ptr->Sd (p_previous_departchoice_file.Sd ());
               Departchoice_ptr->Dtr(p_previous_departchoice_file.Dtr());
               Departchoice_ptr->Ddt(p_previous_departchoice_file.Ddt());
               Departchoice_ptr->Ibd (p_previous_departchoice_file.Ibd ());
               Departchoice_ptr->Utility(p_previous_departchoice_file.Utility());
               Departchoice_ptr->Dt (p_previous_departchoice_file.Dt ());
               Departchoice_ptr->Age (p_previous_departchoice_file.Age ());
               Departchoice_ptr->Gender (p_previous_departchoice_file.Gender ());
               Departchoice_ptr->Flexibility (p_previous_departchoice_file.Flexibility
());
               Departchoice_ptr->Retire (p_previous_departchoice_file.Retire ());
               Departchoice_ptr->Flextimework (p_previous_departchoice_file.Flextimework
());
               Departchoice_ptr->Income (p_previous_departchoice_file.Income ());
               Departchoice_ptr->Trippurpose (p_previous_departchoice_file.Trippurpose
());
              Departchoice_ptr->Is_Changed (p_previous_departchoice_file.Is_Changed ());

        if (!departchoice_data.Add ()) {
                              Error ("Adding Record to the pre-depart Data Array");
                       }
               }
       End_Progress ();
       //p_previous_departchoice_file.Close();
       Print (2, "Number of pre-depart Records Read = %d", Progress_Count ());
       return;

mem_error:
       Error ("Insufficient Memory for pre-depart Data");
       return;

}
```

< /Syslib/Include/Departchoice_Data.hpp >

```
//**********************************************************
//     Departchoice_Data.hpp - depart choice classes
//**********************************************************
#ifndef DEPARTCHOICE_DATA_HPP
#define DEPARTCHOICE_DATA_HPP

#include "Static_Service.hpp"
#include "Class_Index.hpp"
#include "Class_Array.hpp"

//---------------------------------------------------------
//     Departchoice_Data class definition
//---------------------------------------------------------
class Departchoice_Data :  public Class2_Index, public Static_Scale
{
public:
       Departchoice_Data (int traveler = 0, int activity = 0);
       virtual ~Departchoice_Data (void)       {}

       int  Traveler (void)              { return (Key1 ()); }
       int  Hhold (void)                 { return (Key1 () / Traveler_Scale ()); }
       int  Person (void)                { return (Key1 () % Traveler_Scale ()); }
       int  Activity (void)              { return (Key2 ()); }
       int  Et (void)                                  { return (et); }
       int  Rt (void)                                  { return (rt); }
       int  Sd (void)                                  { return (sd); }
       int  Dtr (void)                                         { return (dtr); }
       double  Ddt (void)                          { return (ddt); }
       double  Ibd (void)                          { return (ibd); }
       double  Utility (void)            { return (utility); }
       double  Dt (void)                           { return (dt); }

       int Age (void)             { return (age); }
    int Gender (void)         { return (gender); }
    int Flexibility (void)    { return (flexibility); }
    int Retire (void)      { return (retire); }
    int Flextimework (void)   { return (flextimework); }
    int Income (void)      { return (income); }
    int Trippurpose (void)    { return (trippurpose); }
       int Is_Changed (void)  { return (is_changed); }


       void Traveler (int value)          { Key1 (value); }
       void Hhold (int value)             { Key1 (value * Traveler_Scale () + Person
()); }
```

```
	void Person (int value)			{ Key1 (Hhold () * Traveler_Scale () + 
value); }
	void Activity (int value)		{ Key2 (value); }
	void Et (int value)				{ et = value; }
	void Rt (int value)				{ rt= value; }
	void Sd (int value)				{ sd = value; }
	void Dtr  (int value)			{ dtr = value; }
	void Ddt (double value)			{ ddt = value; }
	void Ibd (double value)			{ ibd = value; }
	void Utility (double value)		{ utility = value; }
	void Dt (double value)			{ dt = value; }

    void Age (int value)			{ age = value; }
    void Gender (int value)			{ gender = value; }
    void Flexibility (int value)	{ flexibility = value; }
    void Retire (int value)			{ retire = value; }
    void Flextimework (int value)	{ flextimework = value; }
    void Income (int value)			{ income = value; }
    void Trippurpose (int value)	{ trippurpose = value; }
	void Is_Changed (int value)		{ is_changed = value; }

private:
	int et,rt,sd,dtr,age,gender,flexibility,
retire,flextimework,income,trippurpose,is_changed;
	double ddt,ibd,utility,dt;

};

//---------------------------------------------------------
//	Departchoice_Array class definition
//---------------------------------------------------------
class Departchoice_Array : public Class2_Array
{
public:
	Departchoice_Array (int max_records = 0);

	bool Add (Departchoice_Data *data = NULL)	{ return (Class2_Array::Add 
(data)); }

	Departchoice_Data * New_Record (bool clear = false, int number = 1)
						{ return ((Departchoice_Data *) 
Class2_Array::New_Record (clear, number)); }

	Departchoice_Data * Record (int index)		{ return ((Departchoice_Data *) 
Class2_Array::Record (index)); }
	Departchoice_Data * Record (void)		{ return ((Departchoice_Data *) 
Class2_Array::Record ()); }
```

```
	Departchoice_Data * Get (int key1, int key2)	{ return ((Departchoice_Data *) Class2_Array::Get (key1, key2)); }
	Departchoice_Data * Get (Departchoice_Data *data)	{ return ((Departchoice_Data *) Class2_Array::Get (data)); }

	Departchoice_Data * Get_GE (int key1, int key2)	{ return ((Departchoice_Data *) Class2_Array::Get_GE (key1, key2)); }
	Departchoice_Data * Get_GE (Departchoice_Data *data) { return ((Departchoice_Data *) Class2_Array::Get_GE (data)); }

	Departchoice_Data * Get_LE (int key1, int key2)	{ return ((Departchoice_Data *) Class2_Array::Get_LE (key1, key2)); }
	Departchoice_Data * Get_LE (Departchoice_Data *data) { return ((Departchoice_Data *) Class2_Array::Get_LE (data)); }

	Departchoice_Data * First (void)	{ return ((Departchoice_Data *) Class2_Array::First ()); }
	Departchoice_Data * Next (void)	{ return ((Departchoice_Data *) Class2_Array::Next ()); }

	Departchoice_Data * Last (void)	{ return ((Departchoice_Data *) Class2_Array::Last ()); }
	Departchoice_Data * Previous (void)	{ return ((Departchoice_Data *) Class2_Array::Previous ()); }

	Departchoice_Data * First_Key (void)	{ return ((Departchoice_Data *) Class2_Array::First_Key ()); }
	Departchoice_Data * Next_Key (void)	{ return ((Departchoice_Data *) Class2_Array::Next_Key ()); }

	Departchoice_Data * Last_Key (void)	{ return ((Departchoice_Data *) Class2_Array::Last_Key ()); }
	Departchoice_Data * Previous_Key (void)	{ return ((Departchoice_Data *) Class2_Array::Previous_Key ()); }

	Departchoice_Data * operator[] (int index)	{ return (Record (index)); }
};

#endif
```

< /Syslib/Include/Departchoice_File.hpp >

```
//*********************************************************
//     Departchoice_File.hpp - Departchoice File Input/Output
//*********************************************************
#ifndef DEPARTCHOICE_FILE_HPP
#define DEPARTCHOICE_FILE_HPP
#include "Db_Header.hpp"

//---------------------------------------------------------
//     Departchoice_File Class definition
//---------------------------------------------------------
class  Departchoice_File : public Db_Header
{
public:

       Departchoice_File (Access_Type access = READ, Format_Type format =
DEFAULT_FORMAT, bool notes_flag = false);
       Departchoice_File (char *filename, Access_Type access = READ, Format_Type format
= DEFAULT_FORMAT, bool notes_flag = false);
       virtual ~Departchoice_File (void);

       int Hhold (void)            { Get_Field (hhold, &lvalue); return (lvalue); }
       int Person (void)           { Get_Field (person, &lvalue); return (lvalue); }
       int Activity (void)         { Get_Field (activity, &lvalue); return (lvalue); }
       int Et (void)            { Get_Field (et, &lvalue); return (lvalue); }
       int Rt (void)               { Get_Field (rt, &lvalue); return (lvalue); }
       int Sd (void)            { Get_Field (sd, &lvalue); return (lvalue); }
       int Dtr (void)              { Get_Field (dtr, &lvalue); return (lvalue); }
       double Ddt (void)           { Get_Field (ddt, &dvalue); return (dvalue); }
       double Ibd (void)           { Get_Field (ibd, &dvalue); return (dvalue); }
       double Utility (void) { Get_Field (utility, &dvalue); return (dvalue); }
       double Dt (void)            { Get_Field (dt, &dvalue); return (dvalue); }

    int Age (void)              { Get_Field (age, &lvalue); return (lvalue); }
    int Gender (void)       { Get_Field (gender, &lvalue); return (lvalue); }
    int Flexibility (void)   { Get_Field (flexibility, &lvalue); return (lvalue); }
    int Retire (void)     { Get_Field (retire, &lvalue); return (lvalue); }
    int Flextimework (void)  { Get_Field (flextimework, &lvalue); return (lvalue); }
    int Income (void)     { Get_Field (income, &lvalue); return (lvalue); }
    int Trippurpose (void)   { Get_Field (trippurpose, &lvalue); return (lvalue); }
       int Is_Changed (void) { Get_Field (is_changed, &lvalue); return (lvalue); }


       void Hhold (int value)             { Put_Field (hhold, value); }
       void Person (int value)            { Put_Field (person, value); }
       void Activity (int value)    { Put_Field (activity, value); }
```

```
        void Et (int value)              { Put_Field (et, value); }
        void Rt (int value)                 { Put_Field (rt, value); }
        void Sd (int value)              { Put_Field (sd, value); }
        void Dtr (int value)         { Put_Field (dtr, value); }
        void Ddt (double value)             { Put_Field (ddt, value); }
        void Ibd (double value)             { Put_Field (ibd, value); }
        void Utility (double value)  { Put_Field (utility, value); }
        void Dt (double value)              { Put_Field (dt, value); }

        void Age (int value)             { Put_Field (age, value); }
    void Gender (int value)              { Put_Field (gender, value); }
    void Flexibility (int value)     { Put_Field (flexibility, value); }
    void Retire (int value)           { Put_Field (retire, value); }
    void Flextimework (int value)    { Put_Field (flextimework, value); }
    void Income (int value)           { Put_Field (income, value); }
    void Trippurpose (int value)     { Put_Field (trippurpose, value); }
        void Is_Changed (int value)      { Put_Field (is_changed, value); }

        virtual bool Create_Fields (void);

protected:
        virtual bool Set_Field_Numbers (void);

private:
        void Setup (void);
        int lvalue;
        double dvalue;
        int hhold,person,activity,et,rt,sd,dtr,ddt,ibd,utility,dt,age,gender,
                flexibility, retire,
                flextimework,income,trippurpose,is_changed;
};

#endif
```