

The Approach-dependent, Time-dependent, Label-constrained Shortest Path Problem and Enhancements for the CART Algorithm with Application to Transportation Systems

Chawalit Jeenanunta

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

Doctor of Philosophy
in
Industrial and Systems Engineering

Hanif D. Sherali, Chair
Antoine G. Hobeika, Co-Chair
Ebru K. Bish
Barbara M. P. Fraticelli
Dusan Teodorovic

May 10, 2004
Blacksburg, Virginia

Keywords: Shortest path, Classification and Regression Tree, Integer Programming

Copyright 2004, Chawalit Jeenanunta

The Approach-dependent, Time-dependent, Label-constrained Shortest Path Problem and Enhancements for the CART Algorithm with Application to Transportation Systems

Chawalit Jeenanunta

(ABSTRACT)

In this dissertation, we consider two important problems pertaining to the analysis of transportation systems. The first of these is an approach-dependent, time-dependent, label-constrained shortest path problem that arises in the context of the *Route Planner Module* of the Transportation Analysis Simulation System (**TRANSIMS**), which has been developed by the *Los Alamos National Laboratory* for the *Federal Highway Administration*. This is a variant of the shortest path problem defined on a transportation network comprised of a set of nodes and a set of directed arcs such that each arc has an associated label designating a mode of transportation, and an associated travel time function that depends on the time of arrival at the tail node, as well as on the node via which this node was approached. The lattermost feature is a new concept injected into the time-dependent, label-constrained shortest path problem, and is used to model turn-penalties in transportation networks. The time spent at an intersection before entering the next link would depend on whether we travel straight through the intersection, or make a right turn at it, or make a left turn at it. Accordingly, we model this situation by incorporating within each link's travel time function a dependence on the link via which its tail node was approached. We propose two effective algorithms to solve this problem

by adapting two efficient existing algorithms to handle time dependency and label constraints: the Partitioned Shortest Path (PSP) algorithm and the Heap-Dijkstra (HP-Dijkstra) algorithm, and present related theoretical complexity results. In addition, we also explore various heuristic methods to curtail the search. We explore an Augmented Ellipsoidal Region Technique (A-ERT) and a Distance-Based A-ERT, along with some variants to curtail the search for an optimal path between a given origin and destination to more promising subsets of the network. This helps speed up computation without sacrificing optimality. We also incorporate an approach-dependent delay estimation function, and in concert with a search tree level-based technique, we derive a total estimated travel time and use this as a key to prioritize node selections or to sort elements in the heap. As soon as we reach the destination node, while it is within some $p\%$ of the minimum key value of the heap, we then terminate the search. We name the versions of PSP and HP-Dijkstra that employ this method as Early Terminated PSP (ET-PSP) and Early Terminated Heap-Dijkstra (ETHP-Dijkstra) algorithms. All of these procedures are compared with the original Route Planner Module within TRANSIMS, which is implemented in the Linux operating system, using C++ along with the g++ GNU compiler.

Extensive computational testing has been conducted using available data from the Portland, Oregon, and Blacksburg, Virginia, transportation networks to investigate the efficacy of the developed procedures. In particular, we have tested twenty-five different combinations of network curtailment and algorithmic strategies on three test networks: the Blacksburg-light, the Blacksburg-full, and the BigNet network. The results indicate that the Heap-Dijkstra algorithm implementations are much faster than the PSP

algorithmic approaches for solving the underlying problem exactly. Furthermore, among the curtailment schemes, the ETHP-Dijkstra with $p=5\%$, yields the best overall results. This method produces solutions within 0.37-1.91% of optimality, while decreasing CPU effort by 56.68% at an average, as compared with applying the best available exact algorithm.

The second part of this dissertation is concerned with the Classification and Regression Tree (**CART**) algorithm, and its application to the Activity Generation Module of TRANSIMS. The CART algorithm has been popularly used in various contexts by transportation engineers and planners to correlate a set of independent household demographic variables with certain dependent activity or travel time variables. However, the algorithm lacks an automated mechanism for deriving classification trees based on optimizing specified objective functions and handling desired side-constraints that govern the structure of the tree and the statistical and demographic nature of its leaf nodes. Using a novel set partitioning formulation, we propose new tree development, and more importantly, optimal pruning strategies to accommodate the consideration of such objective functions and side-constraints, and establish the theoretical validity of our approach. This general enhancement of the CART algorithm is then applied to the Activity Generator module of TRANSIMS. Related computational results are presented using real data pertaining to the Portland, Oregon, and Blacksburg, Virginia, transportation networks to demonstrate the flexibility and effectiveness of the proposed approach in classifying data, as well as to examine its numerical performance. The results indicate that a variety of objective functions and constraints can be readily accommodated to efficiently control the structural information that is captured by the

developed classification tree as desired by the planner or analyst, dependent on the scope of the application at hand.

Acknowledgements

I wish to extend my heartfelt thanks to Dr. Hanif Sherali for the kind assistance and guidance he has provided me throughout my academic life at Virginia Tech. He is truly an excellent instructor as well as researcher, and has always provided me with insightful explanations to guide me to understand the logic underneath the mathematical solution methods. I have learned so much not only from his classes but from working with him. His knowledge and advice were invaluable in my effort to complete this dissertation. Similarly, I would like to express my gratitude to Dr. Antoine Hobeika for his kind support, patience, and guidance over three years. I enjoyed working with him because of his energetic enthusiasm in teaching and apprehending new knowledge. I would also like to acknowledge the other members of my committee, Dr. Ebru K. Bish, Dr. Barbara M. P. Fraticelli, and Dr. Dusan Teodorovic for their assistance, and encouragement.

I thank Ms.Lovedia Cole for her assistance through all the required paperwork. In addition, I would like to thank Boonyarit Intiyot for his patient consultations in regard to coding both in C++ and JAVA. Thanks also to my good friends Nattaporn Chattham, Jonggol Jarupatragorn, Sompid Samipak, and Wisit Singhsomroje for the life journey that we all travel through together. Thanks to my friends in the Thai Student Association for making my life a lot of fun.

I would like to thank my parents, Nikorn and Sureerat Jeenanunta, for their love, emotional, and spiritual support. Thanks to my brother, Weerawat Jeenanunta, who has always been there for me. Finally, I would like to thank all the people in the Development and Promotion of Science and Technology Talents Project for giving me all of these opportunities.

Table of Contents

Chapter 1: Introduction.....	1
1.1 Introduction.....	1
1.2 Description of Research and Organization of Dissertation.....	6
Chapter 2: Literature Review	
2.1 Motivation and Overview.....	8
2.2 Literature review for the Approach-Dependent, Time- Dependent, and Label-Constrained Shortest Path Problem.....	10
2.3 Literature review for the Enhancements for the CART Algorithm	13
Chapter 3: Approach-dependent, Time-dependent, Label- constrained Shortest Path Problem	14
3.1 Introduction.....	14
3.2 Approach-dependent Shortest Path Problem.....	16
3.3 Partitioned Shortest Path (PSP) Implementation	22
3.4 Approach-dependent, Label-constrained Shortest Path Problem	25
3.5 Approach-dependent, Time-dependent, Label-constrained Shortest Path Problem.....	35
3.6 Partitioned Shortest Path (PSP) and Heap-Dijkstra Implementation for the ADTDL CSP problem.....	41
3.7 Curtailment Schemes Based on Estimated Completion Times	50
3.8 Augmented Ellipsoidal Region Technique (A-ERT).....	54
3.9 Computational Comparisons.....	62
3.10 Summary and Conclusions and Recommendation for Future Research.....	79

Chapter 4: Enhancements for the CART Algorithm with Application to Transportation Systems.....	82
4.1 Introduction and Motivation.....	82
4.2 An Overview of TRANSIMS.....	83
4.3 Optimal Pruning Strategy.....	98
4.4 Overall Activity Generator Module.....	107
4.5 Implementation and Results	109
4.6 Summary and Conclusions and Recommendation for Future Research.....	121
References.....	124

List of Figures

Figure 3.1-1: Approach-dependent travel time concept.....	15
Figure 3.2-1: Network for Example 1.....	20
Figure 3.2-2: Expanded Network for Example 1.....	22
Figure 3.3-1: Flow-chart of a PSP implementation for the approach dependent shortest path problem.....	24
Figure 3.4-1: Illustration of G_L for some sample languages L	27
Figure 3.4-2: Illustrations for Example 3.....	29
Figure 3.5-1: An ADTDL graph $G = (N, A, t, \Sigma)$ for Example 4.....	39
Figure 3.6-1: Flow-chart of a PSP implementation for the ADTDL shortest path problem.	43
Figure 3.6-2: Flow-chart of a PSP implementation for the ADTDL shortest path problem (continued).....	44
Figure 3.6-3: Flow-chart of a Heap-Dijkstra implementation for the ADTDL shortest path problem.....	46
Figure 3.6-4: Flow-chart of a Heap-Dijkstra implementation for the ADTDL shortest path problem (continued).....	47
Figure 3.7-1: The angle between the Euclidean line (p,i) and the line (i,j) , $\theta_{(pi,ij)}$	53
Figure 3.8-1: Search region for a freeway and a high-speed metro transit system.....	55
Figure 3.8-2: Flow-chart for finding a high quality transit system within the distance of r	56

Figure 3.8-3: Ellipsoidal region to curtail the network.....	57
Figure 3.8-4: Alternative choice for the freeway path during congestion hours.....	58
Figure 3.8-5: Ellipsoidal Region Technique precluding any path from <i>O</i> to <i>D</i>	60
Figure 3.8-6: Flow-chart of the Distance-Based A-ERT rules to reduce the subnetwork to be used in ADTDLCSP.....	61
Figure 3.9-1: Blacksburg-light network consists of 36 bidirectional links, 31 unidirectional links, and 52 nodes.....	63
Figure 3.9-2: Blacksburg-full network consists of 1251 bidirectional links, 136 unidirectional links, and 1135 nodes	64
Figure 3.9-3: BigNet network consists of 7441 bidirectional links, and 3853 nodes	64
Figure 3.9-4: Example network to calculate the ADLTT summary data.....	67
Figure 3.9-5: Flow-chart of the interactions between the ApproachFormatter, the New Router, and the Microsimulator.....	68
Figure 3.9-6: Pair of origin and destination on Blacksburg-light.	71
Figure 4.2-1: TRANSIMS framework.....	85
Figure 4.2.2-1: Flow-chart for the Tree-Growing Procedure.....	92
Figure 4.2.2-2: Flow-Chart for the Pruning and the Independent Test Sample Approach.....	97
Figure 4.3-1: Illustration for the Proof of Proposition 1.....	103
Figure 4.5-1: A screenshot of CART-VT with the survey data in Excel format	110

Figure 4.5-2: Selection of variables for constructing the CART.....	111
Figure 4.5-3: Pruning method selection for constructing the CART ...	112
Figure 4.5-4: The CART produced by using $N_{\min}=10$ and the multiplier of the total deviance equal to 0.01.....	112
Figure 4.5-5: Various objective functions for optimally pruning the classification tree.....	114
Figure 4.5-6: Various side-constraints for optimally pruning the classification tree	114
Figure 4.5-7: The optimally pruned classification tree for minimizing the number of end nodes subject to this number being at least six.....	115
Figure 4.5-8: The optimally pruned classification tree for minimizing the total number of end nodes while preserving Node 19 and for minimizing the depth of the end nodes while preserving Node 19.....	116
Figure 4.5-9: The optimally pruned classification tree for minimizing the sum over all the end nodes of the ratios of the total number of synthetic households and the survey households that fall into the same end node subject to the total number of end nodes being greater than or equal to five.....	117
Figure 4.5-10: The optimally pruned classification tree for minimizing the total deviance of the new tree.....	118

Figure 4.5-11: The optimally pruned classification tree for minimizing the total deviance of the new tree subject to the ratio between the total number of synthetic households and the survey households that fall into the same end node being less than or equal to seven for all end nodes.....	119
Figure 4.5-12: The optimally pruned classification tree for minimizing the total deviance of the new tree subject to the ratio between the total number of synthetic households and the survey households that fall into the same end node being less than or equal to seven for all end nodes and the depth of each end node being less than or equal to five.....	119
Figure 4.5-13: The optimally pruned classification tree for two objective functions of minimizing the total deviance of the new tree, and maximizing the total number of trips corresponding to the end nodes, where the weight for each objective function was taken as 0.5, subject to the ratio between the total number of synthetic households and the survey households that fall into the same end node being less than or equal to seven for each end node, and with the depth or level of each end node being less than or equal to four.....	120

List of Tables

Table 3.9-1: Approach Dependent Link Travel Times (ADLTT) summary data field records.....	65
Table 3.9-2: Preliminary performance comparison table of the Heap- Dijkstra algorithm, the PSP algorithm, and the ROUTER.	69
Table 3.9-3: Comparison of <i>CPU</i> Times (secs)	72
Table 3.9-4: Comparison of <i>CPU</i> Times (secs) (continued).....	72
Table 3.9-5: Comparison of the <i>Average Solution Quality</i>	73
Table 3.9-6: Comparison of the <i>Average Solution Quality</i> (continued)	73
Table 3.9-7: Comparison of the % of <i>Runs that Yielded Optimal</i> <i>Solutions</i>	73
Table 3.9-8: Comparison of the % of <i>Runs that Yielded Optimal</i> <i>Solutions</i> (continued).....	73
Table 3.9-9: Comparison of the number of plans and the number of transportation plans found by each algorithm.....	73
Table 3.9-10: Comparison of the number of plans and the number of transportation plans found by each algorithm (continued).....	74
Table 3.9-11: Comparison of the <i>CPU</i> Times (secs) for the ETHP- Dijkstra algorithm and the ET-PSP algorithm without the delay estimation function	76
Table 3.9-12: Comparison of the <i>CPU</i> Times (secs) for the ETHP- Dijkstra algorithm and the ET-PSP algorithm without the delay estimation function (continued)	76

Table 3.9-13: Comparison of the <i>Average Solution Quality</i> for the ETHP-Dijkstra algorithm and the ET-PSP algorithm without the delay estimation function.....	76
Table 3.9-14: Comparison of the <i>Average Solution Quality</i> for the ETHP-Dijkstra algorithm and the ET-PSP algorithm without the delay estimation function (continued)	76
Table 3.9-15: Comparison of the % of <i>Runs that Yielded Optimal Solutions</i> for the EThP-Dijkstra algorithm and the ET-PSP algorithm without the delay estimation function.....	76
Table 3.9-16: Comparison of the % of <i>Runs that Yielded Optimal Solutions</i> for the EThP-Dijkstra algorithm and the ET-PSP algorithm without the delay estimation function (continued)...	77

Chapter 1

INTRODUCTION

1.1 Introduction

TRANSIMS, the *Transportation Analysis and Simulation System*, is an integrated system of travel forecasting and simulation models that is designed to provide planners with accurate and comprehensive information on traffic impacts, congestion, and pollution. The Los Alamos National Laboratory (LANL) initiated this effort to develop new transportation and air quality forecasting procedures in response to the Clean Air Act, the Intermodal Surface Transportation Efficiency Act, and other regulations. It is part of the Travel Model Improvement Program sponsored by the U.S. Department of Transportation, the Environmental Protection Agency, and the Department of Energy.

The goal of TRANSIMS is to develop technologies that can be used by transportation planners in any urban environment. TRANSIMS offers transportation planning agencies a robust mechanism for evaluating policies, more detailed vehicle-emission estimates, and improved analysis and visualization capabilities. The underlying philosophy of TRANSIMS is that in order to study the transportation system's performance effectively, one needs to simulate travel in a study area with a rather fine temporal and spatial resolution, based on microanalytic simulation and an activity-based approach for modeling the travel demand.

TRANSIMS starts with a survey database regarding people's activities and the trips they conduct to perform these activities, and then, corresponding to the characteristics of each household, it builds a model for the household activity demand pattern. For each transit activity, TRANSIMS optimally prescribes a route to take corresponding to the specified modes of transportation and time constraints. The model forecasts how changes in transportation policy or infrastructure might affect these activities and trips. TRANSIMS attempts to capture every important interaction between the travel subsystems, such as an individual's activity plans and congestion within the transportation system. For instance, when a trip takes too long, people find alternative routes, or change from a car to train transit, or leave at different times, or decide not to perform a given activity at a given location.

The TRANSIMS framework consists mainly of six modules:

- (1) **Population Synthesizer Module.** This takes two types of census data: land-use data, and network data. Based on this, it generates synthetic households, and creates the identity of individual synthetic travelers. It also locates each household on the network.
- (2) **Activity Generator Module.** This provides a 24-hour period list of activities for each individual synthetic household. These activities are based on demographic surveys and activity surveys that are collected from the real households. These activities include work, shopping, school, etc. In addition, activity times and activity locations are determined for each individual.

- (3) **Route Planner Module.** This develops plans for every individual according to the activity list generated from the Activity Generator. Based on reality, people tend to choose their paths optimally with a specified time limit and mode preference. The Route Planner module computes combined route and mode trip plans to accomplish the desired activities of each individual, such as work, shopping, etc. This is accomplished by solving a defined time-dependent label-constrained shortest path problem.
- (4) **Traffic Microsimulator Module.** This executes travel plans and computes the overall intra- and inter-modal transportation system dynamics. The Microsimulator continuously updates the operational status, including locations, speeds, and acceleration or deceleration of all vehicles throughout the simulation period. The output can provide a detailed, second by second history of every traveler in the system over a 24-hour period. Every motor vehicle in the study area is monitored in this manner to identify traffic congestion and emission concentrations. This module can predict the performance of the transportation system.
- (5) **Emission Estimator Module.** This module estimates the air quality, energy consumption, and pollution emissions. It uses results from the Traffic Microsimulator to predict tailpipe and evaporative emissions for light- and heavy-duty vehicles. These emissions are aggregated and are used to provide overall Regional Air Quality estimates.
- (6) **Selector Module.** This archives internal inconsistencies and selectively feeds back information to the appropriate foregoing modules.

Our focus in this dissertation will be on the Route Planner and the Activity Generator Modules of TRANSIMS. In the Route Planner Module, a key problem that is solved for each traveler is the *time-dependent label-constrained shortest path problem (TDL CSP)*. This is an extension of the traditional shortest path problem that seeks to minimize the total travel time given time-dependent link delay functions. The additional label constraint corresponds to the mode preference of traveling from an origin (O) to a destination (D). For example, if some traveler prefers to walk to the car, use the car as a vehicle to go to the office, and finally walk from the parking lot to the office, TRANSIMS would use the sequence or string of modes specified by “walk-car-walk”. Alternatively, if a traveler prefers to drive to a transit station and then take a transit system such as a train to the destination, the string mode to describe this sequence would be “walk-car-walk-train-walk”. This string of modes is called a label constraint. Note that TRANSIMS inserts the mode “walk” at each transition step. In this dissertation, we also introduce a new feature into TRANSIMS, recognizing that the time spent at an intersection before entering a link would depend on whether we travel straight through the intersection, or make a right turn at it, or make a left turn at it. We denote this problem as the *Approach-Dependent, Time-Dependent, Label-Constrained Shortest Path problem (ADTDL CSP)*. Accordingly, we study how to model and analyze this situation by letting the travel time function for any link depend on the previous link it was approached from at an intersection, in addition to the time-of-day.

The Activity Generator Module assigns an activity list to each synthetic household according to the match between the synthetic population’s demographic characteristics and that of the given household survey via a *Classification and Regression*

Tree (**CART**) approach. This activity list for each member of a synthetic household is developed over a 24-hour horizon, and contains information regarding the activity's origin and destination type (e.g., home, school, workplace, etc.), its priority, starting, ending, and duration time preferences, a preferred travel mode, a vehicle preference (if appropriate), a list of possible physical locations for the activity, and a list of other participants (if the activity is shared). The procedure used to generate the list of activities for each member of a synthetic household involves the following steps. First, skeletal activity patterns are created from the survey households and the CART (Classification and Regression Tree) algorithm is used to build a classification tree based on the household demographic data. Then, given any synthetic household, this is matched with a survey household through the classification tree, in order to generate activity times, durations, and activity locations.

However, the user does not have control or direction on how to achieve some desired objectives in trip or activity generation such as the maximization of the number of trips produced for all households, or maximizing the average number of trips per household that exceed a minimum threshold. Therefore, we will modify the CART algorithm, particularly in the pruning phase, so that a user can avoid extensive trial and error attempts in optimizing desired objectives subject to certain appropriate side-constraints.

1.2 Description of Research and Organization of Dissertation

Our research efforts focus on the ADTDL CSP problem and on enhancing the CART algorithm as introduced above. We propose methods to solve the ADTDL CSP problem using a dynamic programming or Partitioned Shortest Path procedure and a Heap Dijkstra algorithm, and analyze their computational complexity. In addition, we extend the heuristic methods studied by Sherali *et al.* (2003) to improve the run-time of our proposed algorithm. These procedures are tested using real transportation networks pertaining to modeling Portland, OR, and Blacksburg, VA. We also compare the performance of the proposed algorithms against the original Route Planner Module.

To enhance the CART algorithm, we develop an optimal pruning strategy whereby we can optimize various objectives, or goals, subject to specified side-constraints. We present the proposed methodology, establish its theoretical validity, and study its practical efficiency and computational performance using several practical examples based on the available survey data obtained for Portland, OR, and Blacksburg, VA. The proposed methodology provides the user with an interactive tool to control the resulting classification tree through specified objective functions and side-constraints.

The remainder of this dissertation is organized as follows. Chapter 2 provides a review of the related literature. In Chapter 3, we present a discussion of the ADTDL CSP problem as a series of extensions of the traditional shortest path problem. We develop effective solution methods and analyze their computational complexity. We also describe several improvements to the heuristic procedures discussed in Sherali *et al.* (2003) for

curtailing the search in finding good quality solutions. Numerical experiments are conducted using real transportation network test instances to compare the performances of the different proposed algorithms and heuristics.

In Chapter 4, we address the CART algorithmic procedure that is used in TRANSIMS to generate the classification trees for categorizing data. An optimal pruning procedure is proposed that can handle several practical objective functions and side-constraints while developing the classification tree. We conduct extensive computational tests using available real survey data pertaining to Portland, Oregon, and Blacksburg, Virginia. Related computational results are presented to demonstrate the flexibility and effectiveness of the proposed approach in classifying data, as well as to examine its numerical performance.

Chapter 2

LITERATURE REVIEW

2.1 Motivation and Overview

As discussed in Chapter 1, the Route Planner Module of TRANSIMS uses a special algorithm to find a path for each trip that minimizes the total travel time, given that the travel duration on each link is time-dependent, and that the sequence of link labels on these routes must conform with a specified permissible label-string constraint. In traditional shortest path problems, we assume a constant travel time. However, in TRANSIMS, we define the time delay for each link to be a function of the arrival time at that link. If a traveler departs home at 8.00 am to go to the office, this trip could be longer than that when leaving home at 7.00 am. The additional label constraint corresponds to the mode preference of traveling from an origin to a destination. For example, if some traveler prefers to walk to the car, use the car as a vehicle to go to the office, and finally walk from the parking lot to the office, TRANSIMS would use the sequence or string of modes specified by “walk-car-walk”. This string of modes is called a label constraint. In addition, the time spent at the intersection before entering a link could depend on whether we travel straight through the intersection, or make a right turn at it, or make a left turn at it. Accordingly, we study how to model this situation by incorporating within the travel time function for any link a dependence on the previous link via which this was approached at the intersection. Additionally, due to the large number of trips that need to

be computed within TRANSIMS, we need to employ heuristic techniques to curtail the search network in order to reduce the overall computational time required.

In TRANSIMS, the Activity Generator Module takes household demographic survey data and household activity survey data as its input to generate the list of activities for each member in the synthetic population. The household demographic survey contains information about the characteristics of each household member in the survey sample of households. This data is used for matching with the demographic characteristics of the synthetic households. The activity survey is a representative sample of the household activities, including event-participation information for each household member. The Activity Generator Module assigns an activity list to each synthetic household according to the match between the synthetic population's demographic characteristics and that of the household survey via a Classification and Regression Tree (CART) approach. The CART algorithm produces a classification of household characteristics based on households' travel behavior. However, the user does not have control or direction on how to achieve some desired objectives in trip or activity generation such as the maximization of the number of trips produced for all households or maximizing the average number of trips per household that exceed a minimum threshold. Therefore, we will modify the CART algorithm, particularly in the pruning phase, so that the user can avoid extensive trial and error attempts in optimizing desired objectives subject to specified side-constraints.

2.2 Literature review for the Approach-Dependent, Time-Dependent, and Label-Constrained Shortest Path Problem

Mendelzon *et al.* (1995) prove that the labeled constrained shortest path problem is NP-hard. Romeuf (1988) shows that any shortest path problem with additional constraints in a finite A-automaton may be transformed to a usual shortest path problem using some product automaton, for which Dijkstra's algorithm can then be applied. Here, a finite automaton is defined by events, actions and state transitions. A finite A-automaton is a finite automaton with discrete states A-sets. Barrett *et al.* (1998) use nondeterministic finite automata (NFA) to construct a composite graph that consists of all possible combinations of paths that satisfy the specified mode strings or label constraints for the trip from the origin node to the destination node. An NFA is a five-tuple $(S, \Sigma, \delta, s_0, F)$, where S is a finite nonempty set of possible states, Σ is the *input alphabet*, δ is the *state transition function* that maps $S \times \Sigma$ to the set of subsets of S , $s_0 \in S$ is the *initial state*, and $F \subseteq S$ is the set of *accepting (final permissible)* states. For the given graph $G(N, A)$, where N is the set of nodes and A is the set of arcs, and the given transition graph G_L , where the nodes represent suitable labels, and the arcs represent permissible transitions among the labels, Barrett *et al.* define two corresponding NFA. The product of these two NFAs provides a composite graph G^* . They then apply Dijkstra's shortest path algorithm or any other algorithm for the usual shortest path problem to the composite graph G^* to find a label-constrained shortest path.

Instead of constructing the full composite graph *a priori*, Sherali *et al.* (2003) develop an effective method for implicitly working with the composite graph. This

framework is based on the partitioned shortest path algorithmic scheme of Glover *et al.* (1985 a, b) and its dynamic programming (DP) interpretation by Sherali (1991). Sherali *et al.* prove that under the consistency or First-In-First-Out (FIFO) assumption, the complexity of their algorithm is polynomial of order $O(|N|m|A|)$ where $|N|$ is the number of nodes in G , $|A|$ is the number of arcs in G , and m is the maximum number of possible nodes from G_L that could occur at any defined stage. However, in the case of inconsistent or non-FIFO delays, the algorithm is pseudopolynomial of complexity $O(K\eta m|A|)$ where $\eta(\leq T)$ is the maximum number of distinct values of times for which it is possible to visit any node within the given horizon interval $[0, T]$, (assuming integer data), and $K(\leq T)$ is a bound on the number of stages in the DP process.

In addition, Sherali *et al.* (2003) also propose four heuristic curtailing schemes to reduce the search effort. These techniques focus the search to proceed toward the destination from the origin while avoiding the exploration of paths that deviate beyond a different boundary and do not utilize certain beneficial links. Each of these heuristic schemes is based on a particular weight parameter, β_i , applied to a delay-estimation function for each arc from any node i to the destination. The algorithm also imposes a limitation on the acceptable travel time to the destination. These four heuristic curtailing schemes are as follows. (1) **Standard Base-Case**, where the $\beta_i = 1$ for each node i ; (2) **Network Sectioning Technique**, where the network is partitioned into three sections and β_i depends on the particular section that contains node i ; (3) **Level-Based Technique**, where β_i varies according to the depth of the node i from the origin node, and (4) **Ellipsoidal Region Technique**, where β_i is zero if node i lies within a defined ellipsoid whose foci are the origin and the destination nodes, and is infinite otherwise. In

addition, the Ellipsoidal Region Technique also includes the key freeway sections and two additional ellipsoids whose major axes are oriented along the line from the origin node to the entrance of the freeway for one, and along the line from the destination node to the freeway exit for the other. Effectively, this technique considers all paths that lie within the three ellipsoidal regions and the region including the freeway section, and applies the proposed time-dependent label-constrained shortest path algorithm to this network.

The computational results reported for these curtailing heuristic techniques show that the Ellipsoidal Region Technique provides the best compromise between the solution quality and the computational effort. This technique finds solutions within 7% of optimality and saves 33.57 % CPU time with respect to the exact method. In addition to the Ellipsoidal Region Technique, Sherali *et al.* replace two additional ellipsoids with rectangular regions. The quality of solution improves to within 6.7% of optimality, but the CPU effort reduces only by 27.53% of that required by the exact method. The amount of saving in CPU time is critical due to the large number of trips that need to be computed within TRANSIMS.

This curtailing technique could be extended by including the search for any nearby high-speed corridors and transit systems that could help avoid bottlenecks within the transportation system on the search path. If there is an entrance/transit-stop close by the origin, the algorithm could search for and include the corresponding transit routes within a certain corridor along with nearby exit/transit-stops leading to the destination. In the sequel, we shall propose such an idea on how to improve the Ellipsoidal Region

Technique by using the available information to the extent possible. The intent here is to reduce the CPU effort while obtaining near-optimal solutions.

For further details on this subject, we refer the reader to Ahuja *et al.* (1993), Hart *et al.* (1968), Sedgewick and Vitter (1986), as well as to the related discussion in Chapter 3.

2.3 Literature review for the CART Algorithm

Breiman *et al.* (1984) introduced the Classification and Regression Trees (CART), which is an innovative methodology for the analysis of large data sets through a binary partitioning procedure. The CART algorithm first finds a maximal tree by a tree growing step, and then prescribes the final tree via a pruning step. In this context, Taylor and Silverman (1993) propose new splitting or branching criteria and Mola and Siciliano (1997) propose fast splitting algorithms. Cappelli *et al.* (1998) propose an alternative pruning procedure based on an impurity complexity measure. Further details and illustrations of the CART algorithm are provided in the corresponding Chapter 4 that pertains to this topic.

Chapter 3

THE APPROACH-DEPENDENT, TIME-DEPENDENT, LABEL-CONSTRAINED SHORTEST PATH PROBLEM

3.1 Introduction

Consider a graph $G(N,A)$ comprised of a set of nodes N and a set of directed arcs A . Each arc (i, j) has an associated label designating a mode of transportation, and an associated travel time function that depends on the time of arrival at the tail node i , as well as on the link (u, i) along which the node i was approached. The lattermost feature is a new concept injected into the time-dependent, label-constrained shortest path problem discussed by Sherali *et al.* (2003), and is used to model turn-penalties in transportation networks. For example, referring to Figure 3.1-1, the travel along link (i, j) might follow an approach to node i from any of the (reverse star) nodes u_1, u_2 , or u_3 . The time spent at the intersection i before entering link (i, j) would depend on whether we travel straight through the intersection, or make a right turn at it, or make a left turn at it, as the case would be when approaching i from u_1 , or u_2 , or u_3 , respectively. Accordingly, we model this situation by incorporating within the travel time function on link (i, j) a dependence on the link via which node i was approached.

Note that we could have modeled this situation by defining suitable additional nodes and edges as illustrated in Figure 3.1-1 (b) with obvious reformulation details. However, this would tend to blow up the size of the network (imagine several approach-

dependent nodes in the forward star of i as well) and would be inconvenient by requiring a modification of some given, possibly complex, transportation network representation. Hence, we choose to equivalently handle this situation by assuming an approach-dependent travel time function.

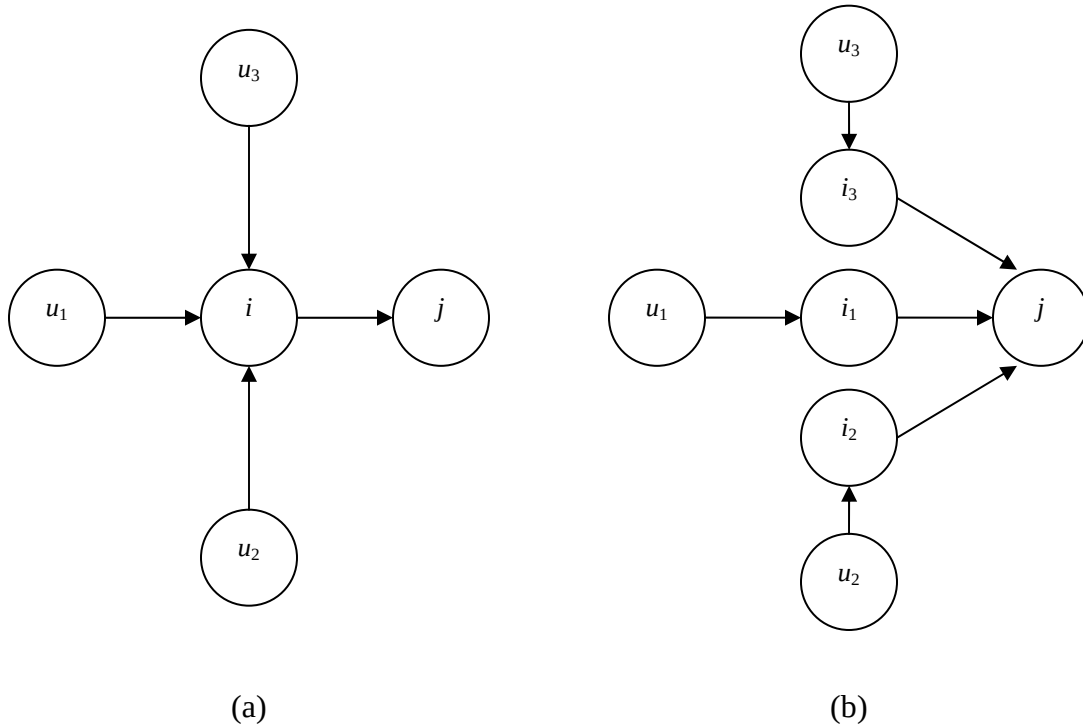


Figure 3.1-1: Approach-dependent travel time concept.

Given an origin node O and a destination node D , the problem is to find a shortest path from O to D that conforms with some specified admissible string of labels. Typically, we might also be given some upper bound T on the permissible travel time from O to D , whereby, the computation of a shortest path from O to D could be aborted once it is provably determined that there does not exist a path for which the total travel time is less than or equal to T .

3.2 Approach-dependent Shortest Path Problem

To begin our discussion, let us first consider the case when the link travel times are approach-dependent, but we do not have any time dynamic aspects, or specified label constraints, or any upper bound T on permissible travel times. Also, while transportation travel times would typically be nonnegative, in order to consider a more general case wherein such a shortest path problem might be defined with respect to mixed-sign reduced cost functions in a decomposition framework, we permit negative function values. However, we assume that the network is devoid of any negative cost circuits. Let us refer to such a problem as an *Approach-dependent Shortest Path Problem*.

Note that in an overall transportation network, there might be only a few nodes i of the type illustrated in Figure 3.1-1, for which the emanating edges have travel time functions that depend on the edge along which i was approached. Let $N_A \subseteq N$ represent such “approach-dependent” nodes, and let us denote $\overline{N}_A \equiv N - N_A$. Also, for any nodes $i \in N$, let $FS(i) = \{j : (i, j) \in A\}$ and $RS(i) = \{j : (j, i) \in A\}$ respectively define the *forward star* and the *reverse star* of i . Without loss in generality, we will assume that $FS(D) = \emptyset$ and $RS(O) = \emptyset$. Given any node $p \in N_A$, we will *conceptually* create clones p_q of node p for each $q \in RS(p)$ in order to represent an approach to p via the edge (q, p) . Accordingly, we will designate the travel time to traverse link (p, j) , for any $j \in FS(p)$, given that node p was approached via link (q, p) , for $q \in RS(p)$, by $t_{p_q j}$. For $p \in \overline{N}_A$, the corresponding travel time function will be denoted simply by

$t_{pj}, \forall j \in FS(p)$. (For convenience or uniformity in notation, we will also sometimes refer to a node $p \in \overline{N}_A$ as p_q where $q = \emptyset$ (i.e., $p_{\emptyset} \equiv p$ for $p \in \overline{N}_A$).)

Let us now describe a dynamic programming (DP) routine to find a shortest path from O to D in such an approach-dependent network. For simplicity in notation, let us assume that $O \equiv 1$ and that $D \equiv n$, where $n=|N|$. (Actually, the prescribed methodology will also find a shortest path from node 1 to all the other nodes in N .) Following this description, we will recast this dynamic programming procedure into the framework of a modified *Partitioned Shortest Path* (PSP) algorithm following Sherali's (1991) analysis that establishes the equivalence of the PSP algorithm due to Glover *et al.* (1985a) with a DP scheme for the usual static shortest path problem.

Toward this end, consider the following definitions of stages, states, decisions, and the (forward) recursive equation for the DP routine. The number of stages (which are implicitly created) will be no more than the maximum number of edges in a simple path from node 1 to any other node (bounded above by $(n-1)$). The overall state-space is designated by

$$S = \{i_{\emptyset} \equiv i, \forall i \in \overline{N}_A, \text{ and } i_j, \forall i \in N_A, \text{ and } j \in RS(i)\}. \quad (1)$$

We will also maintain a shortest path label $w_{i_j}, \forall i_j \in S$, that are initialized as $w_1 = 0$ and $w_{i_j} = \infty$, otherwise. Furthermore, following a forward recursive scheme, the states S_k at Stage k will represent the possible set of states that we could have

transitioned to at this stage, and the decision set D_k at Stage k will represent the possible set of states from which this transition could have occurred.

Specifically, these sets are defined recursively as follows.

Stage 1:

$$D_1 \equiv \{1\} \tag{2a}$$

$$S_1 = \left\{ j \text{ for each } j \in FS(1) \cap \overline{N}_A, j_1 \text{ for each } j \in FS(1) \cap N_A \right\}. \tag{2b}$$

Stage $k, k \geq 2$:

$$D_k = \{p_q \in S_{k-1} : w_{p_q} \text{ was improved at Stage } k-1\} \tag{3a}$$

$$S_k = \bigcup_{p_q \in D_k} \left\{ j \text{ for each } j \in FS(p) \cap \overline{N}_A, j_p \text{ for each } j \in FS(p) \cap N_A \right\}. \tag{3b}$$

If $D_k = \emptyset$, we terminate the process. Otherwise, the recursive equation used for computing a shortest path estimate at Stage k to State j_r , designated $f_k^*(j_r)$, is given by

$$f_k^*(j_r) = \underset{\substack{p_q \in D_k : \\ (p, j) \in A \text{ and} \\ p=r \text{ if } r \neq \emptyset}}{\text{minimum}} \{t_{p_q j} + f_{k-1}^*(p_q)\}, \forall \text{ stages } k \geq 1 \text{ and states } j_r \in S_k, \tag{4}$$

where $f_0^*(1) \equiv 0$. Letting $d_k^*(j_r)$ be an optimal decision from D_k in (4), we also execute the following:

$$\text{If } f_k^*(j_r) < w_{j_r}, \text{ then update } w_{j_r} = f_k^*(j_r) \text{ and let } DOWN(j_r) = d_k^*(j_r), \tag{5}$$

where $DOWN(\cdot)$ records the predecessor index for tracing the shortest path. Note that

the shortest path to node j from node 1 is of value $w_{j_r^*} \equiv \min_{r \in RS(j)} \{w_{j_r}\}$ if $j \in N_A$, and is

given by w_j itself for $j \in \overline{N}_A$, and can be retrieved by tracing back from the state j_{r^*} (where $r^* = \emptyset$ if $j \in \overline{N}_A$) to node 1 using the $DOWN(\cdot)$ labels.

Proposition 1. The foregoing DP routine finds a shortest path from node 1 to each of the other nodes in the network (assuming that no negative cost circuit exists) in polynomial-time having complexity $O(|N|^2|A|)$.

Proof. The legitimacy of the routine follows from the definitions of the stages, states, decisions, and the recursive equation as in (2)-(4), along with Bellman's Principle of Optimality. The latter holds true here since given any stage k and state $j_p \in S_k$, the shortest path from this state onward to any node depends only on the fact that we are at node j that has been approached via the link (p,j) , and not on any other previous states and decisions.

For the complexity argument, observe that the maximum number of stages is $O(|N|)$. Furthermore, at each stage, any node p appears in $p_q \in D_k$ at most $O(|N|)$ times. For each such appearance of p_q , scanning its forward-star to develop the corresponding nodes in S_k takes $O(|FS(p)|)$ time. Performing this for each appearance of p at Stage k takes $O(|N| |FS(p)|)$ time. Summing over all possible nodes p at Stage k gives a complexity of $O(|N| |A|)$ per stage. This yields an overall complexity of $O(|N|^2|A|)$, and the proof is complete. $\square$

Example 1.

Consider the network depicted in Figure 3.2-1, where all links have static travel times as shown against them, except for link (3, 4) for which the travel time is either 2 or 4 units, depending on whether node 3 is approached via link (1, 3) or (2, 3), respectively.

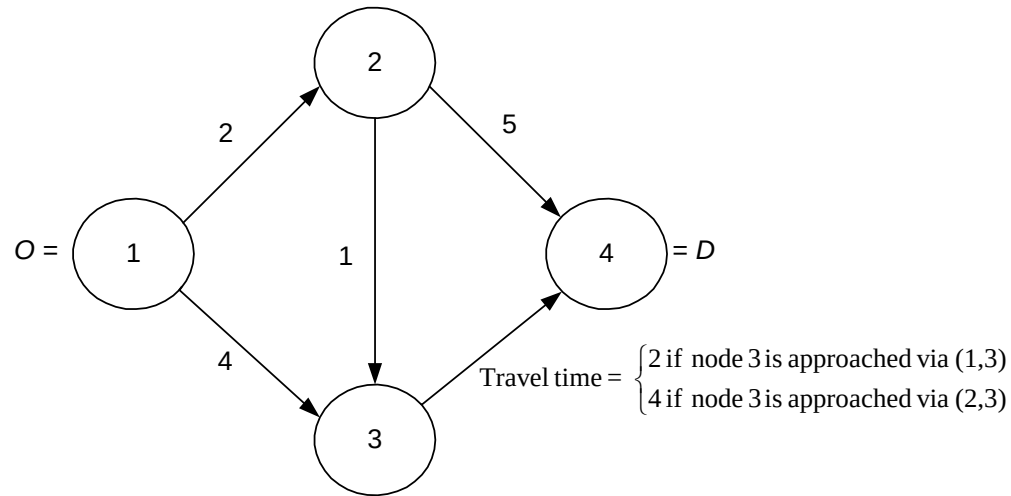


Figure 3.2-1: Network for Example 1.

The stage-wise computations for this example would proceed as follows.

Initialization Note that for this example, we have, $S \equiv \{1_\emptyset \equiv 1, 2_\emptyset \equiv 2, 3_1, 3_2, 4_\emptyset \equiv 4\}$.

We initialize $w_1 = 0$, and $w_{i_j} = \infty$, $\forall i_j \in S, i_j \neq 1_\emptyset$.

Stage 1.

		D_1		
		1	$d_1^*(j_r)$	$f_1^*(j_r)$
$j \in S_1$:	2	2	1	2
	3_1	4	1	4

along with $\{w_2 = 2, DOWN(2) = 1\}$
and $\{w_{3_1} = 4, DOWN(3_1) = 1\}$.

Stage 2.

		D_2			
		2	3_1	$d_2^*(j_r)$	$f_2^*(j_r)$
$j \in S_2$:	3_2	1+2	-	2	3
	4	5+2	2+4	3_1	6

along with
 $\{w_{3_2} = 3, DOWN(3_2) = 2\}$
and $\{w_4 = 6, DOWN(4) = 3_1\}$.

Stage 3.

		D_3			
		3_2	4	$d_3^*(j_r)$	$f_3^*(j_r)$
$j \in S_3$:	4	4+3	-	3_2	7

Note: Since $w_4 = 6 < 7$, no update is performed.

We now terminate since D_4 would be null. The shortest paths from node 1 to all the other nodes are given as follows:

Node 2: $w_2 = 2$; path: $1 \rightarrow 2$.

Node 3: $\min\{w_{3_1}, w_{3_2}\} = w_{3_2} = 3$; path: $1 \rightarrow 2 \rightarrow 3$.

Node 4: $w_4 = 6$; path: $1 \rightarrow 3 \rightarrow 4$, where 3 is represented here by 3_1 .

Remark 1. It is interesting to note here that the shortest path to node 4 passes through node 3, but its sub-path to node 3 is not the shortest path to node 3. Hence, in traversing to node 4 from node 1, it is better for us to take a somewhat longer path $1 \rightarrow 3$ to the intermediate en route node 3, as opposed to the shortest path $1 \rightarrow 2 \rightarrow 3$ to node 3, in order to avoid a relatively greater turn-penalty associated with the latter path in comparison with the former.

Remark 2. It is instructive to also note that this DP process implicitly views the network of Figure 3.2-1 as its expanded counterpart given in Figure 3.2-2 without actually requiring such an expansion *a priori*, which can get cumbersome for more complete networks.

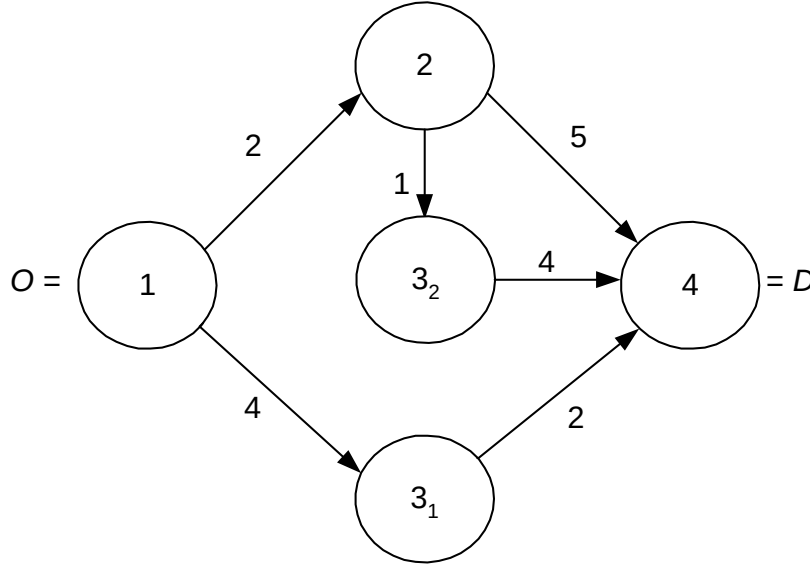


Figure 3.2-2: Expanded Network for Example 1.

3.3 Partitioned Shortest Path (PSP) Implementation

As discussed in Sherali (1991) for the case of the usual static shortest path problem, the foregoing type of a dynamic programming routine can be equivalently implemented via a *partitioned shortest path* (PSP) list structure and labeling scheme. Here at each stage k , the list NOW_k represents D_k and the list $NEXT_k$ in essence builds up the scan-eligible states in S_k , where we set $D_{k+1} = S_k$ at the following stage. Also, the computations in the DP table occur column wise as we scan the forward stars of the nodes corresponding to the states in $D_k = NOW_k$. However, for effectiveness, whenever the label w_{j_r} of a state j_r is revised at any stage, while j_r is still a member of NOW_k that has not as yet been scanned, we can use this updated label at this stage itself when we select to scan j_r (i.e., via the forward star of j), rather than wait to do this at the

subsequent stage. A flow-chart for this procedure is given in Figure 3.3-1. Its worst-case complexity remains the same as that for the DP procedure, i.e., $O(|N||A|^2)$.

Example 2.

Let us illustrate the PSP algorithm of Figure 3.3-1 for the example of Figure 3.2-1.

$k=1$.

$$\begin{bmatrix} NOW_1 \\ NEXT_1 \end{bmatrix} = \begin{bmatrix} \{1\} \\ \{\emptyset\} \end{bmatrix} \rightarrow \begin{bmatrix} \{\emptyset\} \\ \{2, 3_1\} \end{bmatrix} \quad \text{with } \{w_2 = 2, DOWN(2) = 1\} \text{ and } \{w_{3_1} = 4, DOWN(3_1) = 1\}.$$

$k=2$.

$$\begin{bmatrix} NOW_2 \\ NEXT_2 \end{bmatrix} = \begin{bmatrix} \{2, 3_1\} \\ \{\emptyset\} \end{bmatrix} \rightarrow \begin{bmatrix} \{3_1\} \\ \{3_2, 4\} \end{bmatrix} \rightarrow \begin{bmatrix} \{\emptyset\} \\ \{3_2, 4\} \end{bmatrix} \quad \text{along with } \{w_{3_2} = 3, DOWN(3_2) = 2\} \text{ and } \{w_4 = 7, DOWN(4) = 2\} \text{ when scanning } FS(2), \text{ being revised to } \{w_4 = 6, DOWN(4) = 3_1\} \text{ when scanning } FS(3) \text{ for } 3_1 \in NOW_2.$$

$k=3$.

$$\begin{bmatrix} NOW_3 \\ NEXT_3 \end{bmatrix} = \begin{bmatrix} \{3_2, 4\} \\ \{\emptyset\} \end{bmatrix} \rightarrow \begin{bmatrix} \{3_2\} \\ \{\emptyset\} \end{bmatrix} \rightarrow \begin{bmatrix} \{\emptyset\} \\ \{\emptyset\} \end{bmatrix} \Rightarrow \text{stop}.$$

The shortest paths are then traced identically as in Example 1.

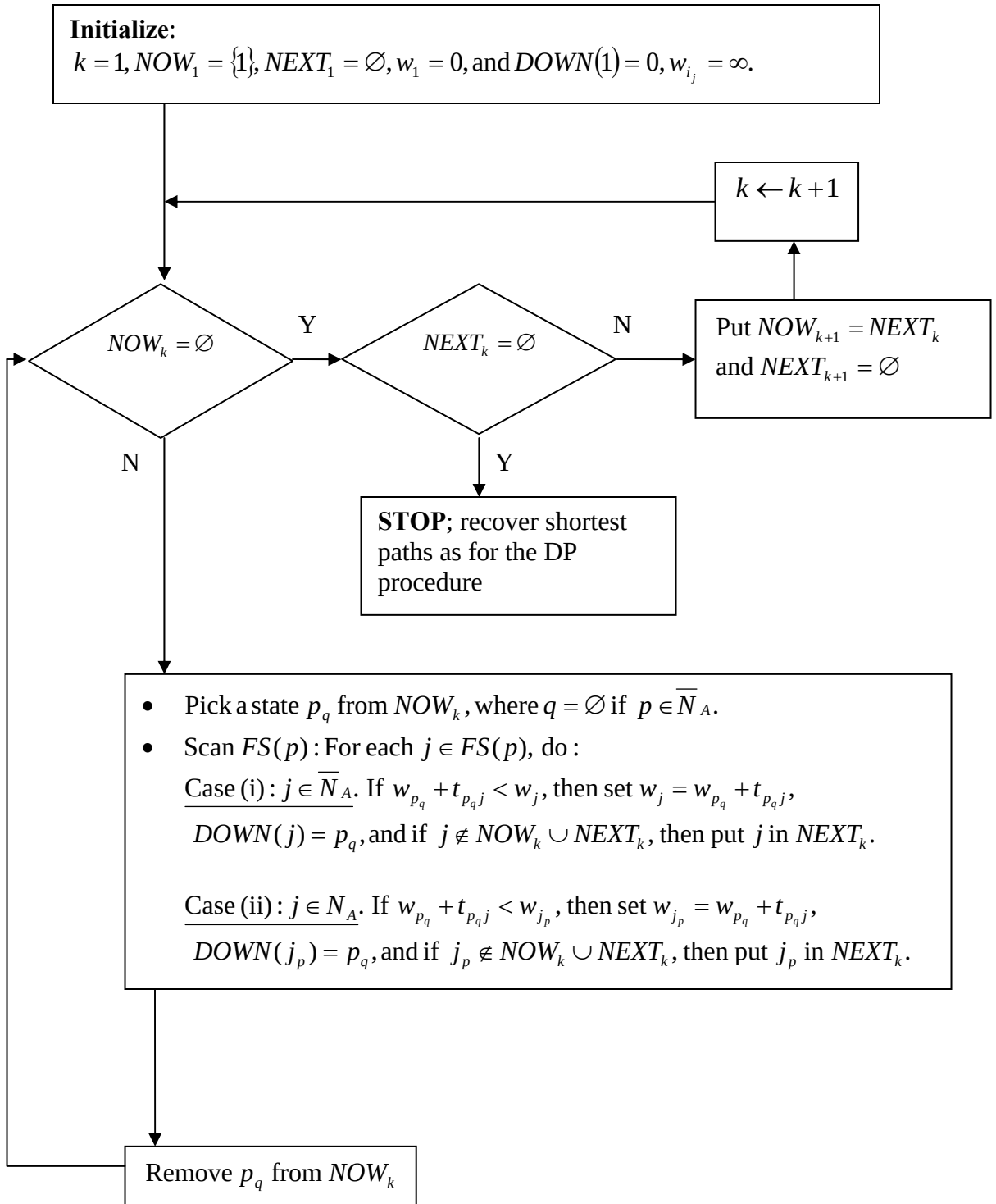


Figure 3.3-1: Flow-chart of a PSP implementation for the approach-dependent shortest path problem.

3.4 Approach-dependent, Label-constrained Shortest Path Problem

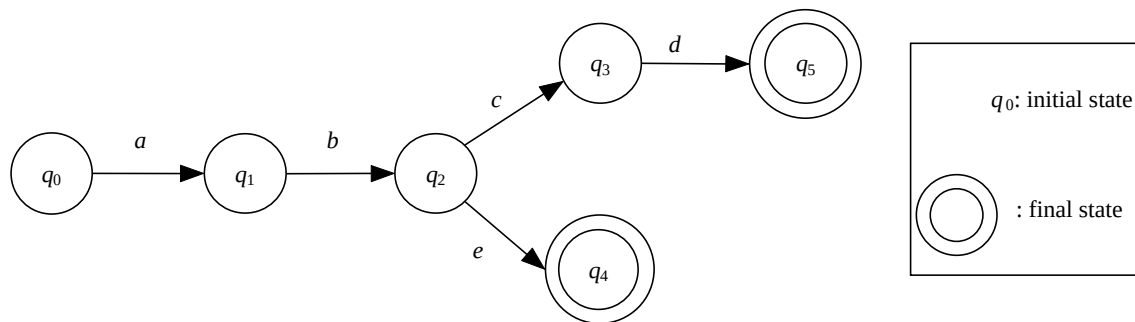
In this section, we extend the approach-dependent concept to the label-constrained shortest path problem, given an origin O and a destination D . In addition to the terminologies from previous section, suppose that each arc is also ascribed a label taken from some *alphabet* Σ , and that we are given a *language* L comprised of *words*, or sequences of alphabets, that constitute acceptable sequences of labels on any selected path $P \in \wp \equiv \{\text{Paths in } G \text{ from node } O \text{ to node } D\}$. Hence, if $\ell(P)$ denotes the word formed by the sequence of labels on the arcs in a path $P \in \wp$, the *Label-constrained Shortest Path Problem* seeks to find a shortest path P^* from O to D from among all paths in the set $\wp \cap \{P : \ell(P) \in L\}$.

Following Sherali *et al.* (2003), examining the admissible label sequences in L , let us define a corresponding *label-transition graph* G_L as follows. The nodes of this graph represent a finite set of states $Q \equiv \{q_r, r = 0, 1, \dots, R\}$ that includes a dummy *initial state* q_0 . The arcs of G_L represent transitions between the states, where each such transition arc has an assigned label $\ell \in \Sigma$, and where for any state (node) q_r , the set of labels associated with the arcs emanating out of q_r are all distinct. Hence, in effect, there exists some *transition function* $\delta : Q \times \Sigma \rightarrow Q$ such that if $(q_{r'}, q_r)$ is an arc in G_L having an assigned label ℓ , then we can write $q_r = \delta(q_{r'}, \ell)$. The construction of this graph G_L in terms of these states (nodes) and labeled transitions (arcs) should be such that for any node q_r , the sequence of labels on any (simple or non-simple) path from q_0 to q_r is part of a legitimate word that is admitted by the specified language. Moreover, there should exist a nonempty set of *final states* $F \subseteq Q$ such that for any $q_r \in F$, the sequence of labels on

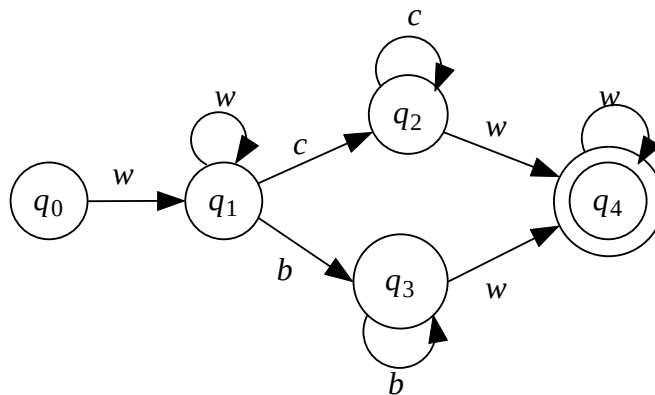
any (simple or non-simple) path from q_0 to q_f is a word in L , and conversely, all realizable words in L are thus represented. We assume that the specified language L is *admissible* if it can be represented by such a label-transition graph G_L . A label-transition graph G_L of this type is also referred to as a *Deterministic Finite Automaton (DFA)* as defined below.

Definition 1: A Deterministic Finite Automaton (**DFA**) is a five-tuple: $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite nonempty set of possible *states*, Σ is the *input alphabet*, δ is the *state transition function* that maps $Q \times \Sigma$ to the set Q , $q_0 \in Q$ is the *initial state*, and $F \subseteq Q$ is the set of *accepting (final permissible)* states.

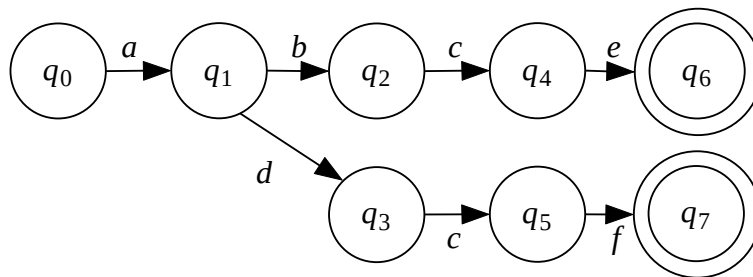
The admissible language L is also called *regular* if it is accepted by a corresponding DFA, which in turn, is associated with a label-transition graph G_L . In order to illustrate the idea behind the label-transition graph G_L , Figure 3.4-1 depicts this graph for some sample languages L . In Figure 3.4-1(a), the language L contains two words that share the first two alphabets, ab . In Figure 3.4-1(b), we use self-loops in G_L to represent the streams of possibly repeated labels in L , where the notation $\bar{\ell}$, for any $\ell \in \Sigma$, denotes a repeated sequences $\ell \dots \ell$ having at least one occurrence of ℓ . Finally, in Figure 3.4-1(c), the language L contains two words having a common alphabet c that appears after some distinct alphabets in the words. Notice how the states (nodes of G_L) are defined in this case to keep the two possible types of occurrences of the label c distinct. In all cases, the graphs G_L correspond to some DFA.



(a) A label-transition graph G_L of $L = \{abcd, abe\}$.



(b) A label-transition graph G_L of $L = \{\overline{wcw}, \overline{wbw}\}$.



(c) A label-transition graph G_L of $L = \{abce, adcf\}$.

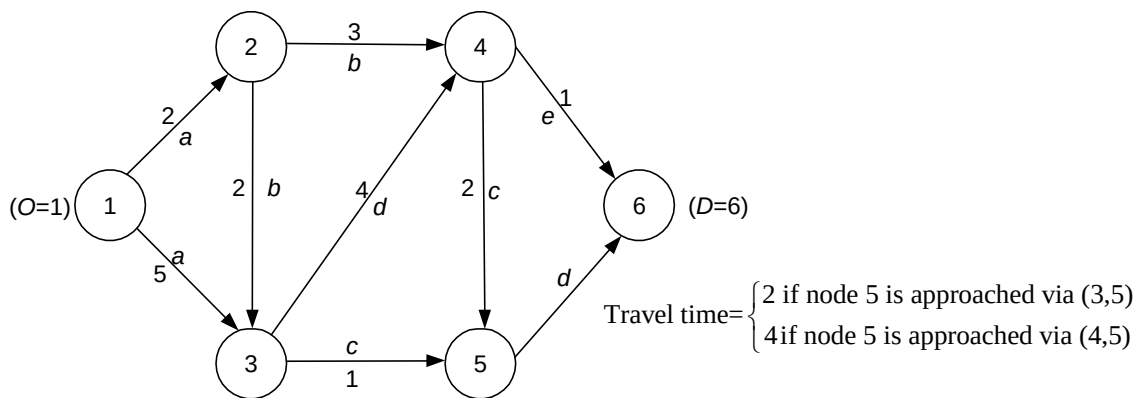
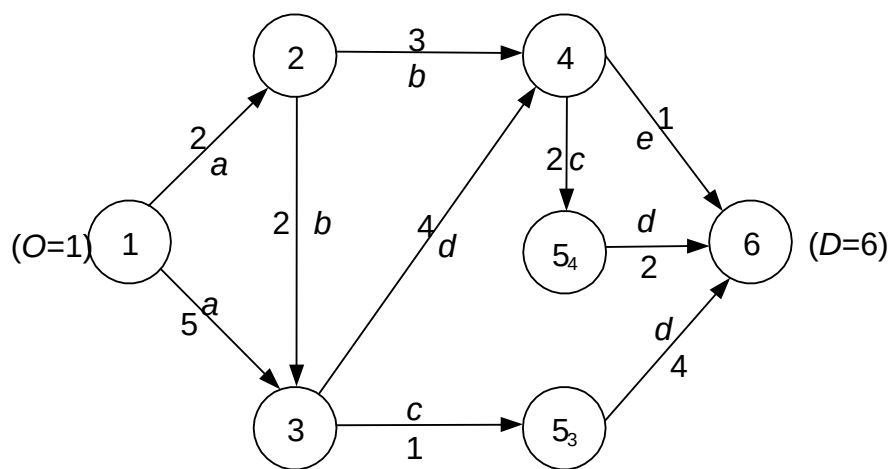
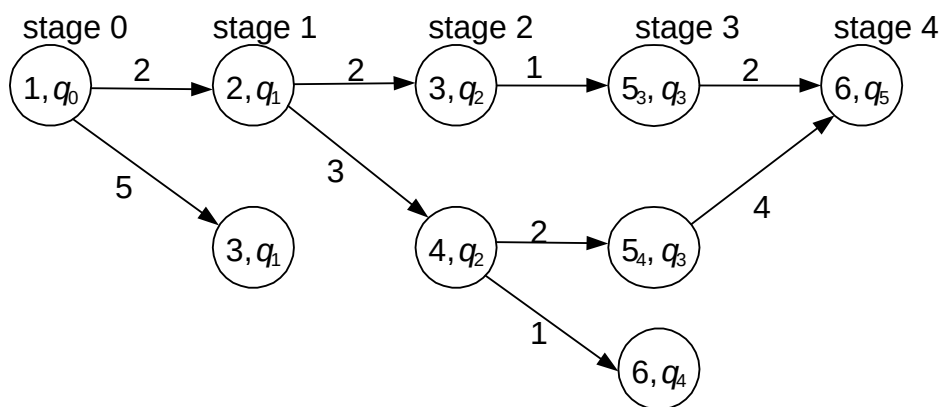
Figure 3.4-1: Illustration of G_L for some sample languages L .

Let us first formally define an *approach-dependent labeled graph* (ADL) $G = (N, A, t, \Sigma)$. This is a directed, labeled graph, where $N = N_A \cup \overline{N}_A$ is a set of nodes as defined previously and A is a set of arcs. Furthermore, the travel time to traverse link (p,j) , for any $j \in FS(p)$, given that node p was approached via link (u,p) , for $u \in RS(p)$, is specified by $t_{p_u j}$. For $p \in \overline{N}_A$, the corresponding travel time function will be denoted simply by t_{pj} , $\forall j \in FS(p)$. Moreover, the labels on the arcs of G are drawn from an alphabet Σ .

Example 3.

Consider the ADL graph illustrated in Figure 3.4-2(a), having static travel times and labels as shown against them, except for link (5,6) for which the travel time is either 2 or 4 units, depending on whether node 5 is approached via link (3,5) or (4,5), respectively. Suppose that we are required to find a shortest path from node $O=1$ to $D=6$ subject to the constraint that the corresponding label sequence should belong to the language $L = \{abcd, abe\}$, where the associated label-transition graph G_L is depicted in Figure 3.4-1(a). The *state-space* is designated by Equation (1) and all the states at stage k correspond to nodes that are reachable from node O in k steps. For the corresponding (conceptually) expanded graph of Figure 3.4-2(a), as shown in Figure 3.4-2(b), we would have the following states for each of the possible five stages.

		stages k					
		0	1	2	3	4	5
states		1	2	3	4	5 ₄	6
			3	4	5 ₄	6	
				5 ₃	5 ₃		
					6		

(a) An ADL graph $G=(N, A, t, \Sigma)$.(b) A conceptually expanded ADL graph $G=(N, A, t, \Sigma)$.(c) Composite graph G^* .**Figure 3.4-2:** Illustrations for Example 3.

We define the *number of stages* K as follows:

$$K = \min\{ UB, \text{length of the maximum admissible string in } L, \\ \text{maximum number of steps from } O \text{ to } D \text{ possible in } G\}, \quad (6)$$

where UB is some finite practical upper bound on the number of stages to be considered.

Using the ADL graph G and the label-transition graph G_L , we can *conceptually* construct a *composite graph* G^* having a stage-wise structure as follows. At stage 0, G^* has the single node (O, q_0) . Then, progressing inductively, given the construction of G^* up to some stage $k \in \{0, 1, \dots, K-1\}$, we construct the nodes of G^* at stage $k+1$ along with the corresponding arcs leading to these nodes from stage k using the following process. For each node $(i'_j, q_{r'})$ of G^* at stage k (where $j = \emptyset$ if $i' \in \bar{N}_A$), if there exists an arc (i', i) in G having a label ℓ such that for some state q_r in G_L we have that $q_r = \delta(q_{r'}, \ell)$, then if we have Case (i): $i \in \bar{N}_A$, we include the node (i, q_r) in G^* at stage $k+1$, and we construct an arc from $(i'_j, q_{r'})$ to (i, q_r) in G^* having a cost of $t_{i'_j i}$; on the other hand, if we have Case (ii): $i \in N_A$, then we include the node $(i_{i'}, q_r)$ in G^* at stage $k+1$, and we construct an arc from $(i'_j, q_{r'})$ to $(i_{i'}, q_r)$ in G^* having a cost of $t_{i'_j i}$. By construction, the graph G^* is acyclic and for any path in G^* from (O, q_0) at stage 0 to (i_j, q_r) at stage k , we have a corresponding path in G , where the associated sequence of labels constitutes a part of an admissible word in L . Moreover, any simple path in G^* from (O, q_0) to (D, q_f) , where $q_f \in F$, corresponds to a path in G having a sequence of labels that constitutes an admissible word in L .

For the case of Example 3, Figure 3.4-2(c) illustrates the composite graph G^* that would be constructed. We can now find a shortest path from node (O, q_0) to (D, q_f) in

G^* using any standard algorithm for computing shortest paths such as Dijkstra's algorithm (see Bazaraa *et al.*, 1990, for example).

To present how this problem can be solved using a dynamic programming (DP) routine without an explicit construction of the composite graph G^* , suppose that $O \equiv 1$ and that $D \equiv n$, where $n=|N|$. The state-space S is defined as the set of nodes in the composite graph G^* , so that we need to find a shortest path from node $(1, q_0)$ to (n, q_f) $\forall q_f \in F$. We will maintain shortest path labels $w_{(i_j, q_r)}, \forall (i_j, q_r) \in S$, which are initialized as $w_{(1, q_0)} = 0$ and $w_{(i_j, q_r)} = \infty$, otherwise. The set of states S_k at stage k and the decision set D_k are defined recursively as follows.

Stage 1:

$$D_1 \equiv \{(1, q_0)\} \quad (7a)$$

$$S_1 = \left\{ \begin{array}{l} (i, q_r) \text{ for each } i \in FS(1) \cap \bar{N}_A \text{ where the label on} \\ \text{arc } (1, i) \text{ in } G \text{ is } \ell \text{ such that } q_r = \delta(q_0, \ell); \text{ along with} \\ (i_1, q_r) \text{ for each } i \in FS(1) \cap N_A \text{ where the label on} \\ \text{arc } (1, i) \text{ in } G \text{ is } \ell \text{ such that } q_r = \delta(q_0, \ell) \end{array} \right\}. \quad (7b)$$

Stage $k, k \geq 2$:

$$D_k = \{(i_j, q_r) \in S_{k-1} : w_{(i_j, q_r)} \text{ was improved at stage } k-1\} \quad (8a)$$

$$S_k = \bigcup_{(i'_j, q_{r'}) \in D_k} \left\{ \begin{array}{l} (i, q_r) \text{ for each } i \in FS(i') \cap \bar{N}_A \text{ where the label on} \\ \text{arc } (i', i) \text{ in } G \text{ is } \ell \text{ such that } q_r = \delta(q_{r'}, \ell); \text{ along with} \\ (i'_j, q_r) \text{ for each } i \in FS(i') \cap N_A \text{ where the label on} \\ \text{arc } (i', i) \text{ in } G \text{ is } \ell \text{ such that } q_r = \delta(q_{r'}, \ell) \end{array} \right\}. \quad (8b)$$

If $D_k = \emptyset$, we terminate the process. Otherwise, the recursive equation used for computing a shortest path estimate at stage k to state (i_p, q_r) , designated $f_k^*(i_p, q_r)$, where $p = \emptyset$ if $i \in \overline{N}_A$, is given by

$$f_k^*(j_r) = \underset{\substack{p_q \in D_k: \\ (p, j) \in A \text{ and} \\ p=r \text{ if } r \neq \emptyset}}{\text{minimum}} \{t_{p_q j} + f_{k-1}^*(p_q)\}, \quad \forall \text{ stages } k \geq 1 \text{ and states } j_r \in S_k, \quad (9)$$

where $f_0^*(1, q_0) \equiv 0$. Letting $d_k^*(i_p, q_r)$ be an optimal decision from D_k in (9), we also execute the following:

$$\text{If } f_k^*(i_p, q_r) < w_{(i_p, q_r)}, \text{ then update } w_{(i_p, q_r)} = f_k^*(i_p, q_r) \text{ and let } \text{DOWN}(i_p, q_r) = d_k^*(i_p, q_r). \quad (10)$$

For the destination node $D \equiv n$, where we can assume that $n \in \overline{N}_A$, we have that the shortest path value from $O \equiv 1$ to $D \equiv n$ is given by

$$\min_{q_f \in F} w_{(n, q_f)} \equiv w_{(n, q_f^*)}, \text{ say,} \quad (11)$$

where the corresponding shortest path can be traced backwards by following the $\text{DOWN}(\cdot)$ labels starting at the node/state (n, q_f^*) at the appropriate stage where $w_{(n, q_f^*)}$ was realized.

The stage-wise computation for Example 3 would proceed as follows.

Initialization $S \equiv \{(1, q_0), (2, q_1), (3, q_1), (3, q_2), (4, q_2), (5_4, q_3), (5_3, q_3), (6, q_4), (6, q_5)\}$.

We initialize $w_{(1, q_0)} = 0$, and $w_{(i_j, q_r)} = \infty$, $\forall (i_j, q_r) \in S, (i_j, q_r) \neq (1, q_0)$. (Note that the overall state-space S will be revealed dynamically by the algorithmic process, and that we need not enumerate it at the initialization step; its identification here is only for illustration purposes.)

Stage 1.

	D_1		$d_1^*(i_p, q_r)$	$f_1^*(i_p, q_r)$
	$(1, q_0)$			
$(i_p, q_r) \in S_1:$	$(2, q_1)$	2	$(1, q_0)$	2
	$(3, q_1)$	5	$(1, q_0)$	5

along with $\{w_{(2, q_1)} = 2, DOWN(2, q_1) = (1, q_0)\}$ and $\{w_{(3, q_1)} = 5, DOWN(3, q_1) = (1, q_0)\}$.

Stage 2.

	D_2		$d_2^*(i_p, q_r)$	$f_2^*(i_p, q_r)$
	$(2, q_1)$	$(3, q_1)$		
$(i_p, q_r) \in S_2:$	$(3, q_2)$	2+2	$(2, q_1)$	4
	$(4, q_2)$	3+2	$(2, q_1)$	5

along with $\{w_{(3, q_2)} = 4, DOWN(3, q_2) = (2, q_1)\}$ and $\{w_{(4, q_2)} = 5, DOWN(4, q_2) = (2, q_1)\}$.

Stage 3.

	D_3		$d_3^*(i_p, q_r)$	$f_3^*(i_p, q_r)$
	$(3, q_2)$	$(4, q_2)$		
$(i_p, q_r) \in S_3:$	$(5_3, q_3)$	1+4	$(3, q_2)$	5
	$(5_4, q_3)$	-	$(4, q_2)$	7
	$(6, q_4)$	-	$(4, q_2)$	6

along with $\{w_{(5_3, q_3)} = 5, DOWN(5_3, q_3) = (3, q_2)\}$, $\{w_{(5_4, q_3)} = 7, DOWN(5_4, q_3) = (4, q_2)\}$,

and $\{w_{(6, q_4)} = 6, DOWN(6, q_4) = (4, q_2)\}$.

Stage 4.

		D_4			$d_4^*(i_p, q_r)$	$f_4^*(i_p, q_r)$
		$(5_4, q_3)$	$(5_3, q_3)$	$(6, q_4)$		
$(i_p, q_r) \in S_4$:	$(6, q_5)$	4+7	2+5	-	$(5_3, q_3)$	7

along with $\{w_{(6, q_5)} = 7, DOWN(6, q_5) = (5_3, q_3)\}$.

We now terminate since D_5 would be null. Note here that both q_4 and q_5 are in the set of final states. Thus, from (11), the shortest path from node 1 to node 6 in graph G has length $\min\{w_{(6, q_4)}, w_{(6, q_5)}\} = \min\{6, 7\} = 6 = w_{(6, q_4)}$, which is realized at Stage 3. Following the $DOWN(\cdot)$ labels from Stage 3 backwards, we trace this shortest path as $(1, q_0) \rightarrow (2, q_1) \rightarrow (4, q_2) \rightarrow (6, q_4)$. This yields the corresponding shortest path in G as $1 \rightarrow 2 \rightarrow 4 \rightarrow 6$, following the labels abe , obtained from G_L of Figure 3.4-1(a) corresponding to the sequence of states q_0, q_1, q_2, q_4 in G_L .

We establish the complexity of DP for solving Problem ADLCSP under the practical assumption that $K = O(|N|)$.

Proposition 2. Given an ADL graph G and given the label-transition graph G_L corresponding to the language L , suppose that K given by Equation (6) is of $O(|N|)$. Let m be the maximum number of possible states in G_L that could occur at any stage k (bounded above by the nodes of G_L). Then the dynamic programming routine proposed for solving ADLCSP is of complexity $O(m|N|^2|A|)$.

Proof. The number of stages enumerated is $O(|N|)$. For each stage k , any node i of G appears in $(i_p, q_r) \in S_k$ at most $m|N|$ times. For each such $(i_p, q_r) \in S_k$, scanning the

forward star of i to examine any $j \in FS(i)$ for which arc (i,j) in G has a label ℓ such that $q_{r'} = \delta(q_r, \ell)$ takes $O(|FS(i)|)$ time, given the transition function δ . Performing this for each repetition of i at stage k takes $O(m|N||FS(i)|)$ time. Summing this over all possible nodes of G appearing at stage k gives a total complexity of $O(m|N||A|)$ per stage. Hence, the overall process is of complexity $O(m|N|^2|A|)$. $\square$

In the case of non-approach-dependent LCSP as discussed by Sherali *et al.* (2003), the time complexity is $O(m|N||A|)$ which is $O(|N|)$ times lower than ADLCSP. The additional factor of $|N|$ in Proposition 2 is due to the number of times any node i can appear at any stage k with respect to its approach. We will now use the foregoing concepts to formalize this discussion in the context of time-dependent travel times in the next section.

3.5 Approach-dependent, Time-dependent, Label-constrained Shortest Path

Problem

In this section, we now additionally consider the link delays to be time-dependent functions. We will designate the travel time to traverse link (i',i) , for any $i \in FS(i')$, given that node i' was approached at time τ via link (j,i') , for $j \in RS(i')$, by $t_{i',i}(\tau) \geq 0$. For $i' \in \overline{N}_A$, the corresponding travel time function will be denoted simply by $t_{i',i}(\tau), \forall i \in FS(i')$. As mentioned in Sherali *et al.* (2003), the time-dependent travel time functions arise in the context of congestion-related delays at different times of the day. This function might be an analytical statistically determined real valued function

defined over a continuous or a discrete time domain. It can be obtained from the tabulated survey observations or from a dynamic traffic assignment based simulation process (see TRANSIMS (2002a)). In the latter case, the function values might be stored in look-up tables being defined as

$$t_{i',i} : H \equiv \{0, \Delta, 2\Delta, \dots, h\Delta\} \rightarrow \{\Delta, 2\Delta, \dots, h'\Delta\}$$

for some discretized time duration Δ and integer h' , and some suitable large integer h such that the characterization of the delay function beyond the time $h\Delta$ is not of practical interest. Note that to obtain the travel time for any τ where $a\Delta < \tau < (a+1)\Delta$ for some value a , we can estimate this value by interpolating between the corresponding travel times for $a\Delta$ and $(a+1)\Delta$.

Let us also specify:

$$\begin{aligned} T = & \text{some upper bound on the total delay of an acceptable path in the} \\ & \text{solution to the underlying problem.} \end{aligned} \tag{12}$$

As discussed in Sherali *et al.* (2003), this additional parameter controls the degree of exploration of the network. It might be user-defined or could be taken as the total delay for some known admissible $O - D$ path. It also induces an upper bound UB on the number of stages as defined in Equation (6).

Similar to the foregoing section, let us define an *approach-dependent and time-dependent labeled graph* (**ADTDL**) $G = (N, A, t, \Sigma)$ as a directed, labeled graph, where $N = N_A \cup \bar{N}_A$ is a set of nodes as defined previously and A is a set of arcs such that the labels associated with each of the arcs of G are drawn from an alphabet Σ . The travel time function, t , is time-dependent as defined at the beginning of this section. We are also given an admissible regular language L based on which we can construct a

corresponding label-transition graph G_L . The ADTDL CSP problem then seeks to find a time-dependent shortest path P^* in G from O to D among all such paths $P \in \wp$, the set of paths in G from O to D , for which $\ell(P) \in L$, where $\ell(P)$ is the sequence of alphabet labels along the path P .

In this context, the proposed algorithm employs time-expanded states (i_j, t, q_r) , for each node $i \in N_A$, $j \in RS(i)$, or $i \in \overline{N_A}$ with $j = \emptyset$, each time $t \in H$, and each state $q_r \in Q$, where t is the time of visitation of node i_j , and q_r is the label-sequence induced state at which this visitation occurred. As before, instead of explicitly generating the corresponding time-expanded composite graph G^* , which can be memory and time intensive, we dynamically generate a reduced-size time-space network implicitly, while simultaneously finding a solution to the ADTDL CSP problem. These states are effectively the nodes of G^* , that are examined during the algorithmic process over the various stages $k = 1, \dots, K$.

To solve this problem using a similar dynamic programming (DP) routine as before, we maintain the three-tuple states (i_j, t, q_r) , where $j = \emptyset$ for $i \in \overline{N_A}$, and where note that the current shortest path label to this state is given by t itself. We then recursively generate the state set S_k for stage $k \geq 1$, based on the previous state set S_{k-1} as follows, starting with $S_0 \equiv \{(1, 0, q_0)\}$,

$$S_k = \bigcup_{(i'_j, t', q_{r'}) \in S_{k-1}} \left\{ \begin{array}{l} (i, t, q_r) \text{ for each } i \in FS(i') \cap \overline{N_A} \text{ where the label on arc } (i', i) \\ \text{in } G \text{ is } \ell \text{ such that } q_r = \delta(q_{r'}, \ell) \text{ and } t = t' + t_{i'j}(t') < T; \text{ along with} \\ (i'_i, t, q_r) \text{ for each } i \in FS(i') \cap N_A \text{ where the label on arc } (i', i) \\ \text{in } G \text{ is } \ell \text{ such that } q_r = \delta(q_{r'}, \ell) \text{ and } t = t' + t_{i'i}(t') < T. \end{array} \right\}. \quad (13)$$

If $S_k = \emptyset$, we terminate the process. We also maintain a predecessor list $DOWN_k(\cdot)$ for each stage k , where for any $(i_{i'}, t, q_r) \in S_k$, $DOWN_k(i_{i'}, t, q_r)$ denotes the corresponding predecessor $(i'_j, t', q_{r'})$ that defined this state via (13). In addition, we keep the incumbent solution, or the best-known solution, in (k^*, t^*, q^*) .

Example 4.

Consider an *approach-dependent and time-dependent labeled graph* (ADTDL) G shown in Figure 3.5-1 with the admissible language $L = \{abcd, abe\}$, where the corresponding label-transition graph G_L is shown in Figure 3.4-1(a). All the links have constant travel time functions except for link (5,6), and all nodes other than node 5 are approach-independent. Note that the travel time function $t_{5,6}(t') = 8 - t'$ does not satisfy the FIFO assumption. The stage-wise computation for the example is as follows.

Initialization.

$S_0 = \{(1, 0, q_0)\}$, where $O \equiv 1$ and $(k^*, t^*, q^*) = \emptyset$.

Stage=1.

$$S_0 = \{(1, 0, q_0)\} \begin{array}{c} \nearrow \\ \searrow \end{array} \begin{array}{c} \{(2, 2, q_1)\} \\ \{(3, 5, q_1)\} \end{array} = S_1$$

Stage=2.

$$S_1 = \begin{array}{c} \{(2, 2, q_1)\} \\ \{(3, 5, q_1)\} \end{array} \begin{array}{c} \nearrow \\ \searrow \end{array} \begin{array}{c} \{(3, 4, q_2)\} \\ \{(4, 5, q_2)\} \end{array} = S_2$$

Stage=3.

$$S_2 = \left\{ \begin{array}{l} (3,4,q_2) \\ (4,5,q_2) \end{array} \right\} \begin{array}{c} \nearrow \\ \longrightarrow \\ \searrow \end{array} \left\{ \begin{array}{l} (5_3,5,q_3) \\ (5_4,7,q_3) \\ (6,9,q_4) \end{array} \right\} = S_3$$

Since $D=6$ and $q_4 \in F$, we set $T = 9$ and $(k^*, t^*, q^*) = (3, 9, q_4)$.

Stage=4.

$$S_3 = \left\{ \begin{array}{l} (5_3,5,q_3) \\ (5_4,7,q_3) \end{array} \right\} \begin{array}{c} \longrightarrow \\ \longrightarrow \end{array} \left\{ \begin{array}{l} \cancel{(6,12,q_5)} \\ (6,8,q_5) \end{array} \right\} = S_4$$

Since $(6,12,q_5)$ has a total travel time greater than T , we exclude $(6,12,q_5)$ from the set

S_4 . Moreover, since $D=6$ and $q_5 \in F$, we set $T = 8$ and $(k^*, t^*, q^*) = (4, 8, q_5)$.

We stop at Stage 5 because $S_5 = \emptyset$. To retrieve the shortest path, starting at Stage $k^*=4$, and state $(D, t^*, q^*) = (6, 8, q_5)$, we trace back via the predecessor lists in the reverse order: $(6, 8, q_5) \xleftarrow{d} (5_4, 7, q_3) \xleftarrow{c} (4, 5, q_2) \xleftarrow{b} (2, 2, q_1) \xleftarrow{a} (1, 0, q_0)$. Thus, the path $1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 6$ having the label sequence $abcd$ and a delay of $t^* = 8$ solves the given ADTDLCSP problem.

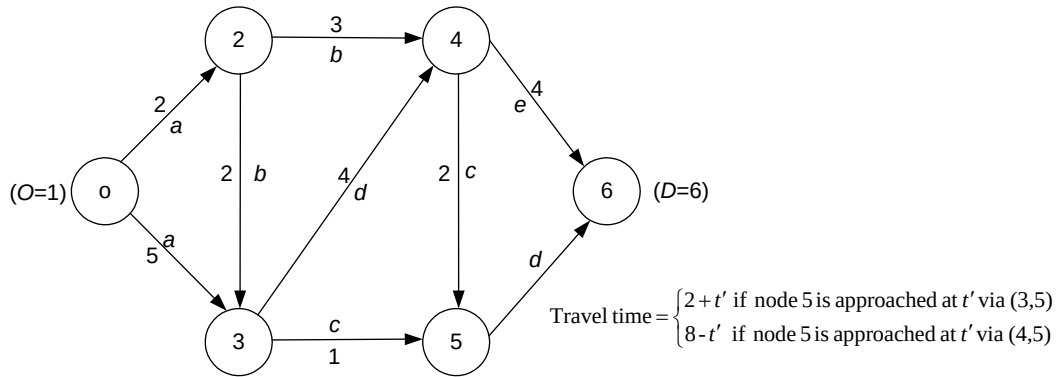


Figure 3.5-1: An ADTDL graph $G=(N, A, t, \Sigma)$ for Example 4.

Remark 1. Note here that this example does not satisfy the *first-in-first-out (FIFO)* or *consistency* assumption as mentioned earlier. The arrival time at node 5 via link (3,5) is 5 which is earlier than arriving at the same node through link (4,5) at time 7. However, it puts us at node 6 at time $5+(2+5) = 12$, while the latter approach puts us at node 6 at time $7+(8-7)=8$, which happens to be earlier than in the former case. Thus, for the general case, we have to maintain all of the possible generated states subject to feasibility with respect to the language L . (An obvious exception is if i corresponds to the final node D .) However, in the case when the consistency assumption holds true, whenever any states (i', t, q_r) are generated at stage k for multiple values of t , we need only retain one of these that corresponds to the smallest value of t . Accordingly, we can modify Equation (13) to eliminate dominated states as in Equation (14) given below.

$$S_k = \bigcup_{(i'_j, t', q_{r'}) \in S_{k-1}} \left\{ \begin{array}{l} (i, t, q_r) \text{ for each } i \in FS(i') \cap \bar{N}_A \text{ where the label on arc } (i', i) \\ \text{in } G \text{ is } \ell \text{ such that } q_r = \delta(q_{r'}, \ell) \text{ and } t = t' + t_{i'j}(t') < T; \text{ along with} \\ (i', t, q_r) \text{ for each } i \in FS(i') \cap N_A \text{ where the label on arc } (i', i) \\ \text{in } G \text{ is } \ell \text{ such that } q_r = \delta(q_{r'}, \ell) \text{ and } t = t' + t_{i'j}(t') < T. \end{array} \right\}. \quad (14a)$$

$$\text{Additionally, } S_k \leftarrow S_k - \{(i', t, q_r) \text{ (possibly, } i' = \emptyset) : \text{there exists} \\ \text{a state } (i', t'', q_r) \in S_k \text{ such that } t'' < t\}. \quad (14b)$$

Proposition 3. Given an ADTDL graph G and given the label-transition graph G_L corresponding to the language L , suppose that we have consistency (FIFO) link delay functions and that K given by Equation (6) is of $O(|N|)$. Let m be the maximum number of possible states in G_L that could occur at any stage k (bounded above by the nodes of G_L). Then the proposed dynamic programming routine for solving ADTDL CSP is of complexity $O(m|N|^2|A|)$.

Proof. Same as that for Proposition 2. □

Proposition 4. Given an ADTDL graph G and given the label-transition graph G_L corresponding to the language L , suppose that we have inconsistency (non-FIFO) link delay functions and assume that we have integer valued delay functions, where η is the maximum number of distinct values of times for which it is possible to visit any node within the interval $[0, T]$ (we can take $\eta = T$). Noting that $K \leq T$ under these assumptions, then the proposed dynamic programming routine for solving ADTDL CSP is of pseudo-polynomial complexity $O(mK\eta|N||A|)$.

Proof. Similar to that for Proposition 2. □

Note here that for both cases, the time complexities are $O(|N|)$ times higher than that of TDL CSP problem as discussed by Sherali *et al.* (2003). This factor is due to the number of times any node i can appear at any stage k with respect to its approach.

3.6 Partitioned Shortest Path (PSP) and Heap-Dijkstra Implementation for the ADTDL CSP problem

As discussed previously, we can equivalently implement the dynamic programming routine for solving ADTDL CSP through a partitioned shortest path (PSP) list structure as shown in Figure 3.6-1. Let us illustrate the PSP algorithm for the problem of Example 4.

Example 5.**k=1.**

$$\begin{bmatrix} NOW_1 \\ NEXT_1 \end{bmatrix} = \begin{bmatrix} \{(1,0,q_0)\} \\ \{\emptyset\} \end{bmatrix} \rightarrow \begin{bmatrix} \{\emptyset\} \\ \{(2,2,q_1),(3,5,q_1)\} \end{bmatrix}$$

with $DOWN_1(2,2,q_1) = (1,0,q_0)$ and $DOWN_1(3,5,q_1) = (1,0,q_0)$.

k=2.

$$\begin{bmatrix} NOW_2 \\ NEXT_2 \end{bmatrix} = \begin{bmatrix} \{(2,2,q_1),(3,5,q_1)\} \\ \{\emptyset\} \end{bmatrix} \rightarrow \begin{bmatrix} \{(3,5,q_1)\} \\ \{(3,4,q_2),(4,5,q_2)\} \end{bmatrix} \rightarrow \begin{bmatrix} \{\emptyset\} \\ \{(3,4,q_2),(4,5,q_2)\} \end{bmatrix}$$

with $DOWN_2(3,4,q_2) = (2,2,q_1)$ and $DOWN_2(4,5,q_2) = (2,2,q_1)$.

k=3.

$$\begin{bmatrix} NOW_3 \\ NEXT_3 \end{bmatrix} = \begin{bmatrix} \{(3,4,q_2),(4,5,q_2)\} \\ \{\emptyset\} \end{bmatrix} \rightarrow \begin{bmatrix} \{(4,5,q_2)\} \\ \{(5_3,5,q_3)\} \end{bmatrix} \rightarrow \begin{bmatrix} \{\emptyset\} \\ \{(5_3,5,q_3),(5_4,7,q_3)\} \end{bmatrix}$$

with $DOWN_3(5_3,5,q_3) = (3,4,q_2)$, $DOWN_3(5_4,7,q_3) = (4,5,q_2)$, and with $(6,9,q_4)$

generated via $(4,5,q_2)$ having $DOWN_3(6,9,q_4) = (4,5,q_2)$, $(k^*,t^*,q^*) = (3,9,q_4)$, and

 $T=9$.**k=4.**

$$\begin{bmatrix} NOW_4 \\ NEXT_4 \end{bmatrix} = \begin{bmatrix} \{(5_3,5,q_3),(5_4,7,q_3)\} \\ \{\emptyset\} \end{bmatrix} \rightarrow \begin{bmatrix} \{(5_4,7,q_3)\} \\ \{\emptyset\} \end{bmatrix} \rightarrow \begin{bmatrix} \{\emptyset\} \\ \{\emptyset\} \end{bmatrix} \Rightarrow \text{stop};$$

where $(6,12,q_5)$ generated via $(5_3,5,q_3)$ is suppressed since $12 > 9$, and $(6,8,q_5)$

generated via $(5_4,7,q_3)$ results in $DOWN_4(6,8,q_5) = (5_4,7,q_3)$, $(k^*,t^*,q^*) = (4,8,q_5)$,

and $T=8$.

The shortest path is now traced as in Example 4.

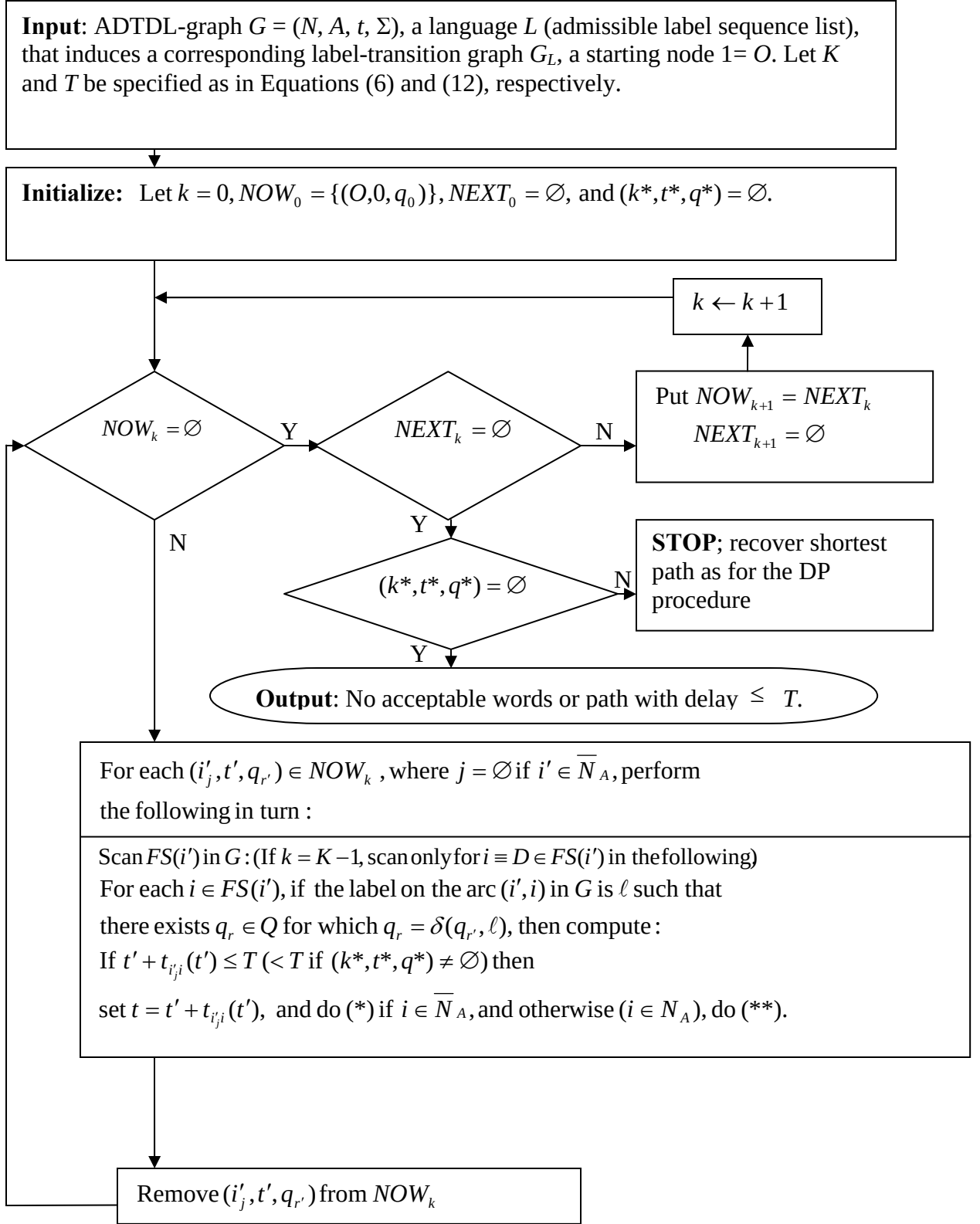


Figure 3.6-1: Flow-chart of a PSP implementation for the ADTDLC shortest path problem.

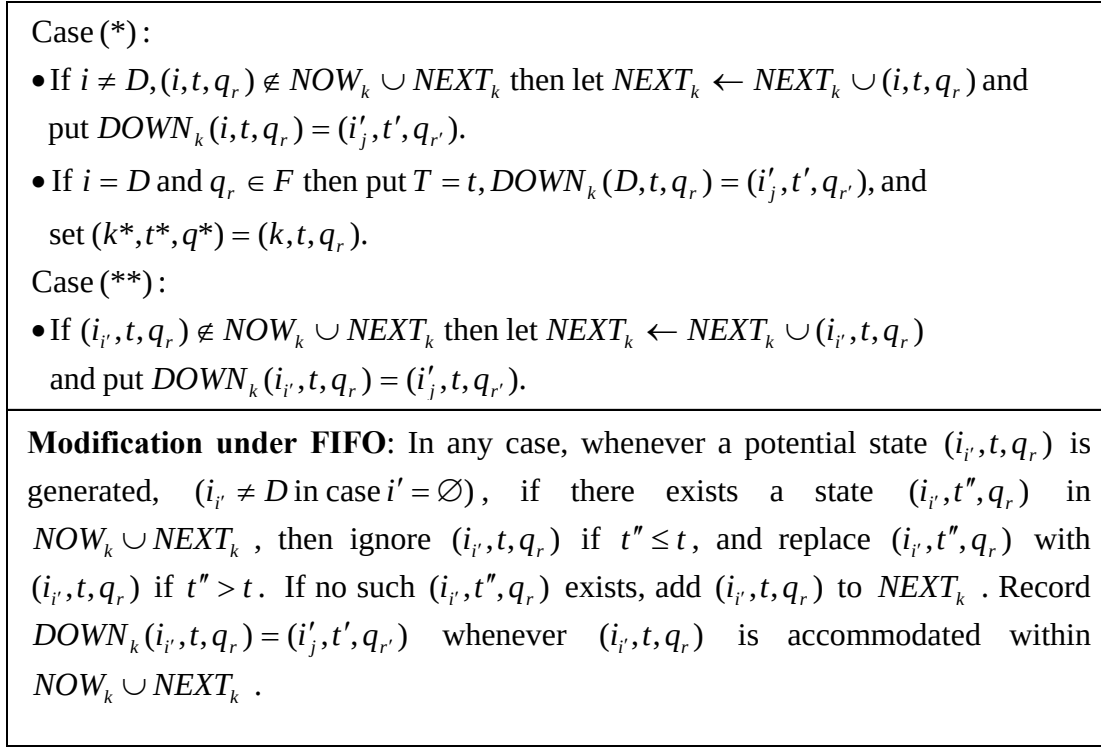


Figure 3.6-2: Flow-chart of a PSP implementation for the ADTDLC shortest path problem (continued).

By using the label-transition graph in the manner discussed previously, we can also essentially implement the traditional Dijkstra's algorithmic scheme. It has been suggested by Ahuja *et al.* (1993) that in Dijkstra's procedure, the bottleneck operation is node selection. Thus, maintaining a sorted list of nodes according to their distance or time labels could reduce the computational effort. The Fibonacci heap and the Radix heap implementation are two alternative methods for improving traditional shortest path algorithms, where the Fibonacci heap implementation achieves the best theoretical complexity of $O(|A| + |N| \log |N|)$. Thus, we will use the Heap-Dijkstra algorithm as an alternative implementation of ADTDLC. Figure 3.6-3 presents this scheme. Here, the various heap operations are defined as follows.

HP is a heap data structure that collects *objects*, each with an associated real number called its *key*, which in our case is the visited time t of each node.

Create-heap(HP) : Creates an empty heap.

Find-min(k, HP) : Finds and returns an *object* k of minimum key.

Insert(k, HP) : Inserts a new *object* k with a predefined key.

Delete-min(k, HP) : Deletes an *object* k of the minimum key.

Empty(HP) : Returns true if heap is empty.

Note that the traditional Dijkstra's algorithm works under the assumption that the arc costs are nonnegative, which is true in our case since the costs correspond to travel times. Also, note that under the consistency assumption, whenever (i_r, t, q_r) is generated for insertion into the heap HP , if there already exists an (i_r, t'', q_r) in HP for any t'' , we can retain only the one that has a smaller time value.

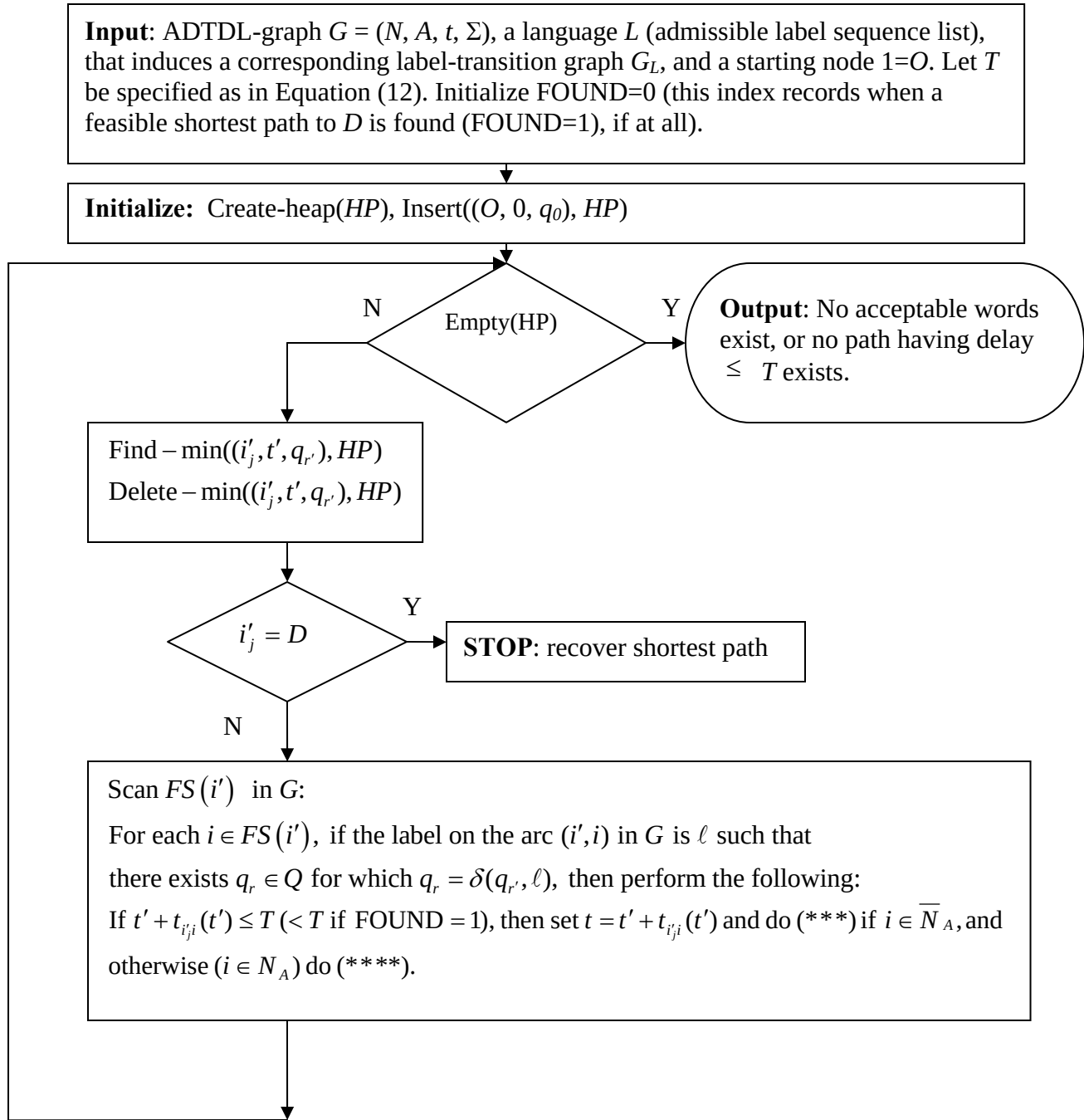


Figure 3.6-3: Flow-chart of a Heap-Dijkstra implementation for the ADTDL shortest path problem.

Case (***) :

- If $i \neq D$, then $\text{Insert}((i, t, q_r), \text{HP})$ and put $\text{DOWN}(i, t, q_r) = (i'_j, t', q_{r'})$.
- If $i = D$ and $q_r \in F$ then put $T = t$, $\text{FOUND} = 1$, $\text{DOWN}(D, t, q_r) = (i'_j, t', q_{r'})$, and $\text{Insert}((D, t, q_r), \text{HP})$.

Case (****) :

- $\text{Insert}((i_{i'}, t, q_r), \text{HP})$ and put $\text{DOWN}(i_{i'}, t, q_r) = (i'_j, t', q_{r'})$.

Figure 3.6-4: Flow-chart of a Heap-Dijkstra implementation for the ADTDLC shortest path problem (continued).

Example 6.

Consider again the problem of Example 4, and let us illustrate the Heap-Dijkstra implementation to solve the problem the given ADTDLCSP problem.

Step 1.

$$HP = [(1, 0, q_0)] \rightarrow \begin{bmatrix} (2, 2, q_1) \\ (3, 5, q_1) \end{bmatrix}$$

with $\text{DOWN}(2, 2, q_1) = (1, 0, q_0)$ and $\text{DOWN}(3, 5, q_1) = (1, 0, q_0)$.

Step 2.

$$HP = \begin{bmatrix} (2, 2, q_1) \\ (3, 5, q_1) \end{bmatrix} \rightarrow \begin{bmatrix} (3, 4, q_2) \\ (4, 5, q_2) \\ (3, 5, q_1) \end{bmatrix}$$

with $\text{DOWN}(3, 4, q_2) = (2, 2, q_1)$ and $\text{DOWN}(4, 5, q_2) = (2, 2, q_1)$.

Step 3.

$$HP = \begin{bmatrix} (3,4,q_2) \\ (4,5,q_2) \\ (3,5,q_1) \end{bmatrix} \rightarrow \begin{bmatrix} (5_3,5,q_3) \\ (4,5,q_2) \\ (3,5,q_1) \end{bmatrix}$$

with $DOWN(5_3,5,q_3) = (3,4,q_2)$.

Step 4.

$$HP = \begin{bmatrix} (5_3,5,q_3) \\ (4,5,q_2) \\ (3,5,q_1) \end{bmatrix} \rightarrow \begin{bmatrix} (4,5,q_2) \\ (3,5,q_1) \\ (6,12,q_5) \end{bmatrix}$$

along with $T=12$ and $DOWN(6,12,q_5) = (5_3,5,q_3)$.

Step 5.

$$HP = \begin{bmatrix} (4,5,q_2) \\ (3,5,q_1) \\ (6,12,q_5) \end{bmatrix} \rightarrow \begin{bmatrix} (3,5,q_1) \\ (5_4,7,q_3) \\ (6,9,q_4) \\ (6,12,q_5) \end{bmatrix}$$

along with $T=9$, $DOWN(5_4,7,q_3) = (4,5,q_2)$ and $DOWN(6,9,q_4) = (4,5,q_2)$.

Step 6.

$$HP = \begin{bmatrix} (3,5,q_1) \\ (5_4,7,q_3) \\ (6,9,q_4) \\ (6,12,q_5) \end{bmatrix} \rightarrow \begin{bmatrix} (5_4,7,q_3) \\ (6,9,q_4) \\ (6,12,q_5) \end{bmatrix}$$

Step 7.

$$HP = \begin{bmatrix} (5_4,7,q_3) \\ (6,9,q_4) \\ (6,12,q_5) \end{bmatrix} \rightarrow \begin{bmatrix} (6,8,q_5) \\ (6,9,q_4) \\ (6,12,q_5) \end{bmatrix}$$

along with $T=8$, and $DOWN(6,8,q_5) = (5_4,7,q_3)$.

Step 8.

$$HP = \begin{bmatrix} (6,8,q_5) \\ (6,9,q_4) \\ (6,12,q_5) \end{bmatrix}$$

At this step, we find that the selected node corresponds to our destination, which is node 6. Thus, we stop our procedure and trace back our shortest path through the predecessor list. The shortest path value is 8, and the labels yield the reverse sequence: $(6,8,q_5) \xleftarrow{d} (5_4,7,q_3) \xleftarrow{c} (4,5,q_2) \xleftarrow{b} (2,2,q_1) \xleftarrow{a} (1,0,q_0)$. Thus, the path $1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 6$ having the label sequence $abcd$ and a delay of $t^* = 8$ solves the given ADTDLCS problem.

The heap implementation of Dijkstra's algorithm, which is shown in Figure 3.6-3, performs the operations Find-min, Delete-min, and Insert at most the number of nodes/states enumerated. Each different heap implementation varies in its complexity of these operations. For the Fibonacci heap data structure, every heap operation is performed in $O(1)$ time except for Delete-min, which requires $O(\log |N_s|)$, where $|N_s|$ is the number of nodes/states. The Fibonacci heap maintains the minimum key so that finding it requires $O(1)$ time. In the ADTDLCS problem, the maximum number of nodes/states that are enumerated is $m|N|^2\eta$. Therefore, the time complexity of the Heap-Dijkstra's implementation with the Fibonacci heap is of $O(|A| + m|N|^2\eta \log(m|N|^2\eta))$, where the factor $|A|$ arises from scanning the forward stars of nodes and the label-setting nature of the algorithm given nonnegative arc costs (see Ahuja *et al.* (1993)).

3.7 Curtailment Schemes Based on Estimated Completion Times

In order to reduce the computational effort, Sherali *et al.* (2003) proposed various heuristic schemes to focus the search network in the direction from O toward the direction of D . These schemes were motivated by the methods of Hart *et al.* (1968) and Sedgewick and Vitter (1986) for finding shortest (distance-based) paths in Euclidean graphs, and are based on estimating a lower bound on the time travel from any intermediate node i to the terminal node D . Denoting $x^i = (x_1^i, x_2^i)^T$ to be the coordinates of node $i \in N$ in some two-dimensional Cartesian space in which the given ADTDL-graph $G = (N, A, t, \Sigma)$ is embedded, a *delay-estimation function* $\mathcal{D}(i, j): N \times N \rightarrow R$ is defined as

$$\mathcal{D}(i, j) = \frac{\|x^i - x^j\|}{v}, \forall i, j \in N, \quad (15)$$

where v is some average estimated velocity of travel over G . Accordingly then, whenever a state (i_r, t, q_r) is identified for inclusion in the *Scan Eligible* (SE) list, which might be S_{k+1} , *NEXT*, or *Heap*, as in the respective procedure of (14), Figure 3.6-1, and Figure 3.6-3, we include (i_r, t, q_r) in *SE* only if in *addition*, we have

$$t + \beta_i \mathcal{D}(i, D) \leq T \quad (16)$$

where β_i , for $i \in N$, is some weighting parameter as prescribed by each of some four heuristic schemes. The delay estimation function $\mathcal{D}(i, D)$ is more likely accurate when i is relatively close to D and is likely to be a weaker lower bound when i is further away from D . The four heuristic schemes that Sherali *et al.* evaluate are (1) **Standard Base-Case**, where the $\beta_i = 1$ for each node i , (2) **Network Sectioning Technique**, where the

network is partitioned into three sections and β_i depends on the particular section that contains node i , (3) **Level-Based Technique**, where β_i varies on the depth of the node i from the origin node, and (4) **Ellipsoidal Region Technique**, where β_i is zero if node i lies within a defined ellipsoid whose focii are the origin and the destination nodes, and is infinite otherwise. The measures used by Sherali *et al.* to compare the relative performances of these algorithms were the *Average Solution Quality* (ASQ), (defined as the travel time obtained from the heuristic method divided by the optimal travel time), the CPU processing time (in seconds on a 450 MHz Pentium II computer with 128 MB of RAM), the percentage of runs that attained an optimal solution, and the average standard deviation of the foregoing values reported across the problem instances. The optimal solution was computed using an exact method. Based on the ASQ value, the best three heuristic methods in order were determined to be:

(1) **Standard Base-Case**,

(2) **Level-Based Technique**, where β_i is

$$\beta_i = \max\{1, 0.9 + 0.5e^{-\lambda h(i)}\}, \forall i \in N, \text{ where } \lambda = \frac{-\log_e(0.2)}{\alpha n} \quad (17)$$

where $h(i)$ is the level away from the origin node O , n is the number of nodes in the problem, and $\alpha=0.25$.

(3) **Ellipsoidal Region Technique** (ERT).

While Method (1) provided the best ASQ values because it is most conservative, it consequently also consumed the greatest CPU times among the four heuristic methods. TRANSIMS generally solves for many trips in each run. Thus, the CPU time is also a

critical issue. Method (3) gave the best compromise between the ASQ values and the CPU times, solving TDL CSP within 7% of optimality and saving 33.57% CPU time, at an average.

Note that a further modification of the algorithms could be achieved by maintaining the *SE* list in the order of the *total estimated travel time* as given by $t + \mathcal{D}_A(i_p, j)$, where we can modify the delay-estimation function to depend on the node p via which i was approached. Let $\theta_{(pi,ij)}$ be the angle between the Euclidean line (p,i) and the line (i,j) as depicted in Figure 3.7-1. We assume that $p \rightarrow i \rightarrow j$ is a *left turn movement*, or a *through movement*, or a *right turn movement*, according as $0 < \theta_{(pi,ij)} \leq 120$, $120 < \theta_{(pi,ij)} \leq 240$, or $240 < \theta_{(pi,ij)} < 360$, respectively. Accordingly, we define the *approach-dependent delay estimation function* as

$$\mathcal{D}_A(i_p, j) = \frac{\|x^i - x^j\|}{v} + \alpha_{i_p j}, \quad \forall i, j \in N, \quad (18)$$

$$\text{where } \alpha_{i_p j} = \begin{cases} \alpha_{i1} & \text{if } p \rightarrow i \rightarrow j \text{ is a through movement} \\ \alpha_{i2} & \text{if } p \rightarrow i \rightarrow j \text{ is a right turn movement} \\ \alpha_{i3} & \text{if } p \rightarrow i \rightarrow j \text{ is a left turn movement} \end{cases}$$

and where $\alpha_{i3} \geq \alpha_{i2} \geq \alpha_{i1} \geq 0$. Based on this estimation function, we adopt the following rule to select a node:

$$(i'_j, t', q_r) \in \operatorname{argmin}\{t + \beta_i \mathcal{D}_A(i_p, D) : (i_p, t, q_r) \in SE\} \quad (19)$$

where *SE* is the list maintained in the heap. Thus, the *key* for the heap in this modified algorithm is the *total estimated travel time* $t + \beta_i \mathcal{D}_A(i_p, D)$. In the case of PSP, when we select nodes from *NOW* according to Equation (19), we could attempt to terminate with the *first time* we label a node as (D, t, q_r) , with $q_r \in F$. Alternatively, instead of

attempting to terminate the algorithm as soon as we label a node as (D, t, q_r) , with $q_r \in F$, we could first compare the corresponding time t with the minimum key currently in the heap, say t' . If $t \leq (1+p)t'$, where $0 \leq p < 1$, then we terminate the search. Otherwise, we put the new labeled node into the heap and continue the search. This is the scheme we implemented. Let us call this early termination criterion-based PSP approach as **Early Terminated PSP** (ET-PSP) algorithm.

In the case of Heap-Dijkstra, we could just as well retain all the nodes generated without loss of effectiveness (instead of tossing out those having *total estimated travel times* that exceed T), while sorting and selecting nodes according to Equation (19) and using the above early termination criterion. Denote this modified Heap-Dijkstra procedure as the **Early Terminated Heap-Dijkstra** (ETHP-Dijkstra) algorithm.

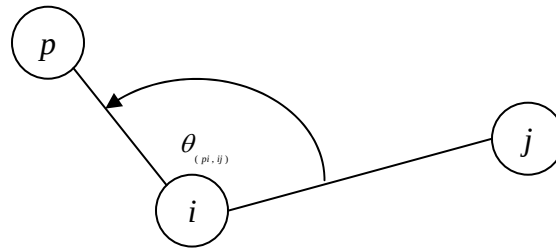


Figure 3.7-1: The angle $\theta_{(pi, ij)}$ between the Euclidean line (p, i) and the line (i, j) .

3.8 Augmented Ellipsoidal Region Technique (A-ERT)

In the original Ellipsoidal Region Technique (ERT), Sherali *et al.* curtail the network to a subset that generally points in the direction from O to D , while including any nearby freeway sections. For any trips that involve relatively large distances, the time that might be taken to traverse through local routes would usually be larger than that using an accessible freeway due to the number of stops made during the trip. Thus, travelers prefer to travel on freeways or to use high-speed metro transit systems or a combination of freeways and local routes. These alternative routes would typically capture an optimal solution. The following discussion explains how to search for such nearby freeway or high-speed transit systems.

Let us define the **subnetwork** R of G as an ADTDL-graph containing the subset of nodes $N_R \subseteq N$ and the subset of arcs $A_R \subseteq A$ such that the algorithm will perform the search on this subnetwork R . For the pair of nodes O and D , we first construct an ellipsoidal region E_1 containing O and D as in the original ERT procedure and initialize R as E_1 .

To search for alternative routes, let us first consider a **metro transit** (M). We begin by growing a search tree from each of O and D using the forward star and reverse star lists, respectively, within some specified **search distance** r , until we obtain a node labeled as a *rail stop*. Denote these nodes as O_M and D_M , respectively, if they exist. Let $M(O_M, D_M)$ represent the metro system section from O_M to D_M . Figure 3.8-1 depicts this concept graphically, where C_1 and C_2 are the circular search regions centered respectively at O and D , and having a radius r . The search distance r can be varied according to the distance between O and D as given by

$$r = \frac{p \|x^D - x^O\|}{3}, \text{ for some } 0 < p \leq 1. \quad (20)$$

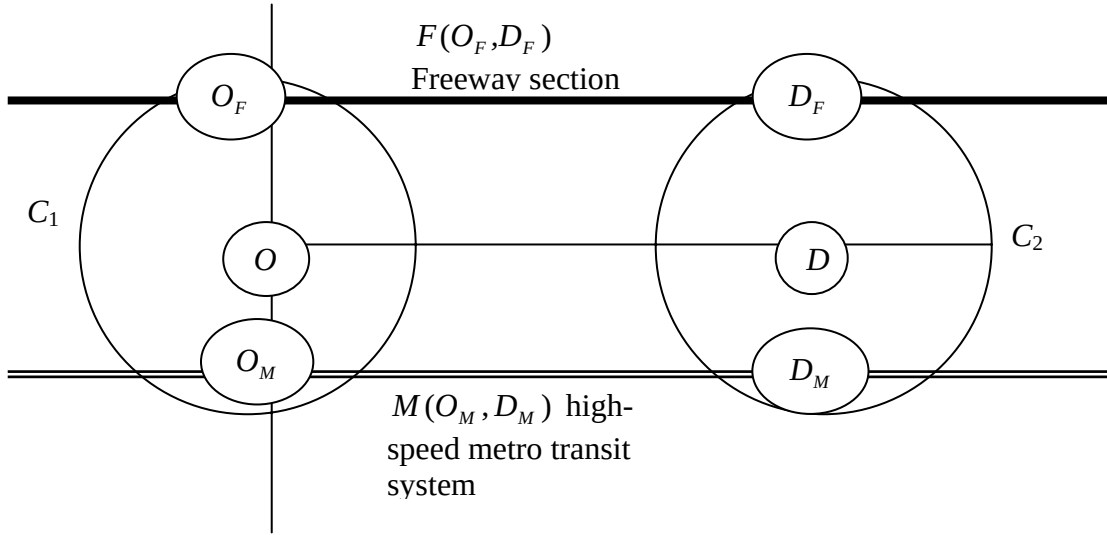


Figure 3.8-1: Search region for a freeway and a high-speed metro transit system.

The subroutine that searches for such alternative routes is described in Figure 3.8-2. After we obtain the nodes O_M and D_M , we then construct the ellipsoidal regions E_2 and E_3 as shown in Figure 3.8-3, and let

$$R \leftarrow R \cup E_2 \cup E_3 \cup M(O_M, D_M). \quad (21)$$

By using the same subroutine of Figure 3.8-2, we can search for a freeway system access for the trip from O to D . Let F represent the identified freeway section having an entrance O_F and an exit D_F that lie within the respective search regions C_1 and C_2 as

Input: ADTDL-Graph $G = (N, A, t, \Sigma)$, a starting node O , a terminal node D , start time t_0 , estimated arrival time T_D , and a search distance r . Let HS be the sorted pair list of entrance and exit nodes of a high-speed transit system within a distance r from O and D , respectively, sorted by the total travel time from O to D using the high-speed transit system between that entrance and exit nodes. Let HP_O be the heap data structure where the objects it collects are nodes and arrival times, and where the key of node i is $\|x^i - x^O\|$, and let HP_D be the heap data structure where the objects it collects are nodes and arrival times, and where the key of node i is $\|x^i - x^D\|$. Let $Type_{HS}$ be the type of the high-speed transit system that we would like to search. Let L_O be the list of nodes.

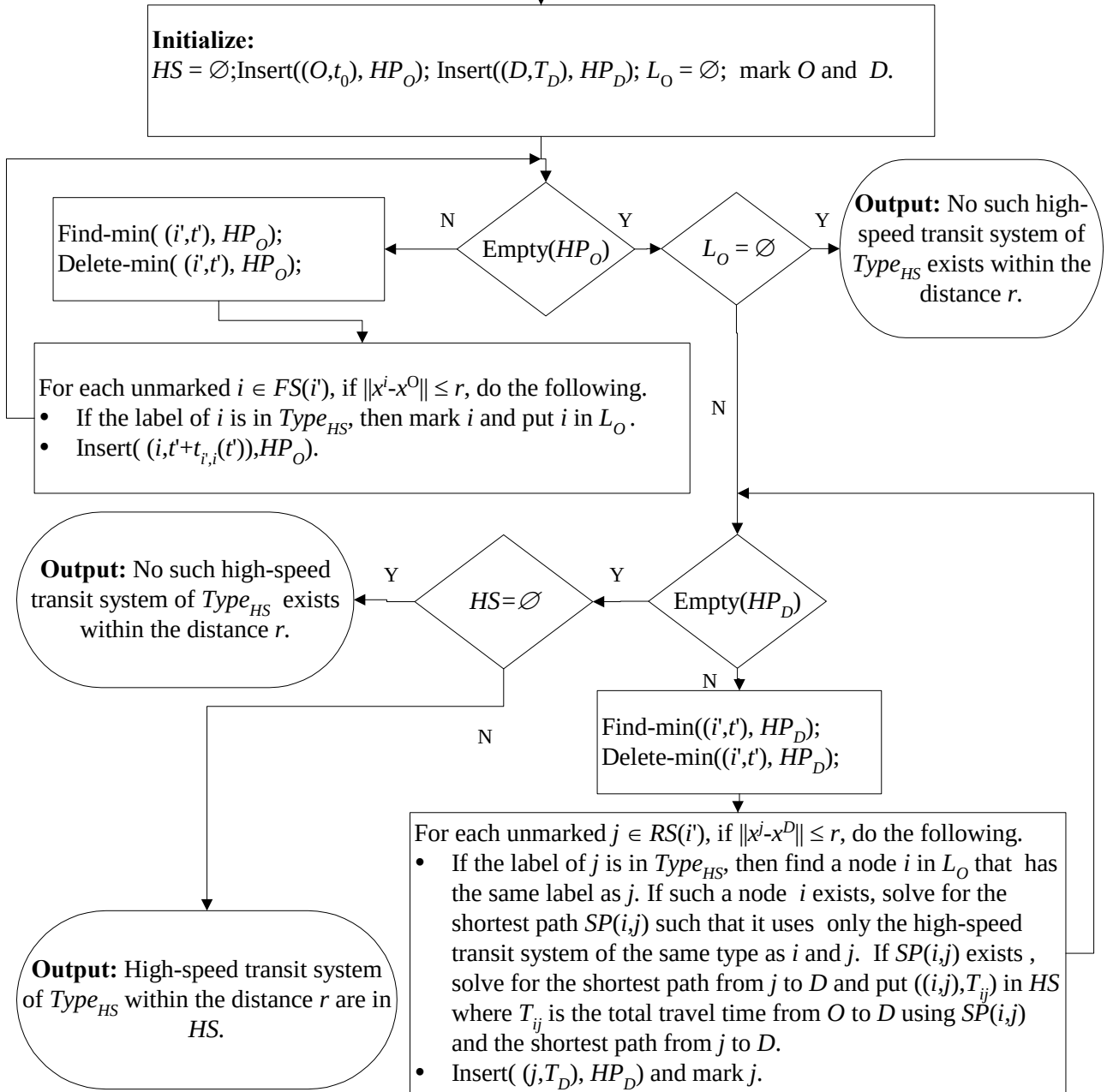


Figure 3.8-2: Flow-chart for finding a high quality transit system within the distance of r .

depicted in Figure 3.8-1. Denote the freeway segment between O_F and D_F as $F(O_F, D_F)$. In case there is no freeway segment that directly connects O_F and D_F , we can use O_F and D_F as the new origin and destination in a nested fashion and solve for the shortest path from O_F to D_F similarly, and use this resultant path as $F(O_F, D_F)$.

Assume that starting at O at $t=0$, the estimated arrival time to O_F is given by $\mathcal{D}(O, O_F)$ as per Equation (15). Let $\mathcal{D}_{true}(O_F, D_F)$ be the actual travel time from O_F to D_F along the freeway section F . If $\mathcal{D}_{true}(O_F, D_F) \leq \mathcal{D}(O_F, D_F) + \Delta$ for some suitable tolerance $\Delta \geq 0$, then construct the ellipsoidal regions E_4 and E_5 as shown in Figure 3.8-3, and let

$$R \leftarrow R \cup E_4 \cup E_5 \cup F. \quad (22)$$

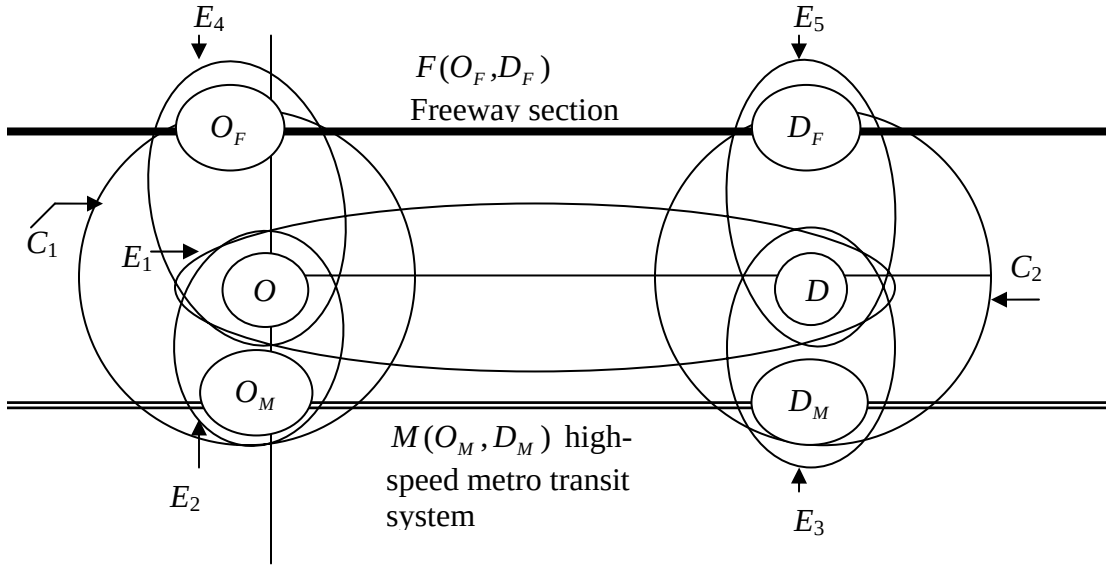


Figure 3.8-3: Ellipsoidal region to curtail the network.

In the case of $\mathcal{D}_{true}(O_F, D_F) > \mathcal{D}(O_F, D_F) + \Delta$ due to congestion, examine the nodal breakpoints along the freeway between O_F and D_F , and identify O_F^+ and D_F^- such that $[O_F^+, D_F^-]$ is the largest contiguous segment of $[O_F, D_F]$ on F such that $\mathcal{D}_{true}(O_F^+, D_F^-) \leq \mathcal{D}(O_F^+, D_F^-) + \Delta$. Define the ellipsoidal regions E_4^{new} and E_5^{new} as shown in Figure 3.8-4, and let $F(O_F^+, D_F^-)$ be the freeway section from O_F^+ to D_F^- . Then replace R by

$$R \leftarrow R \cup E_4^{new} \cup E_5^{new} \cup F(O_F^+, D_F^-). \quad (23)$$

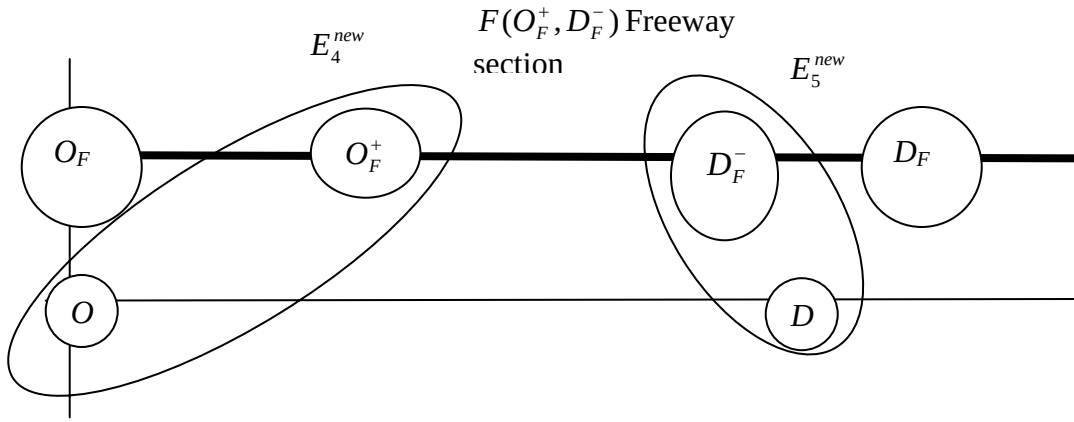


Figure 3.8-4: Alternative choice for the freeway path during congestion hours.

We then solve ADTDL CSP using the proposed algorithms as discussed previously, replacing the original network G with the reduced subnetwork R . Denote this heuristic as an *Augmented Ellipsoidal Region Technique* (A-ERT). In addition, we can search for alternative routes depending on the distance between O and D . The further the distance between O and D , the higher the need for using freeways or metro systems. Thus, we can use a *threshold distance* to determine when to search for the alternative

routes and use these in combination with a localized subnetwork as above, or when to use only such alternative routes.

Let δ_1 and δ_2 be given distance thresholds, where $\delta_1 < \delta_2$. Let $EL(i, j)$ be the subset of the ADTDL-graph G where all nodes and arcs are within a defined ellipsoidal region that encompasses nodes i and j . Define $SP(i, j)$ as the shortest path from node i to j where i and j are the entrance and exit nodes of a high-speed transit system with the restriction that such a path uses only this high-speed transit system. The Distance-Based rules as detailed in Figure 3.8-6 are used to designate the particular reduced subnetwork R to be used in the ADTDL CSP algorithm. These rules are based on the motivation that if the trip distance is short, say within a $\delta_1=5$ miles radius, then the tendency would be to use a local route. If the distance is between 5 miles and, say, $\delta_2=15$ miles, then a combination of local routes and freeway/metro segments might be used. For greater distances, high-speed transit systems such as freeways and rail transit will most likely be the best choices. Denote this heuristic as the *Distance-Based A-ERT*.

In sparse networks, however, such an A-ERT procedure could curtail the network to a point where there is no path from O to D as depicted in Figure 3.8-5. In such a case, we could run ETHP-Dijkstra without the curtailing technique.

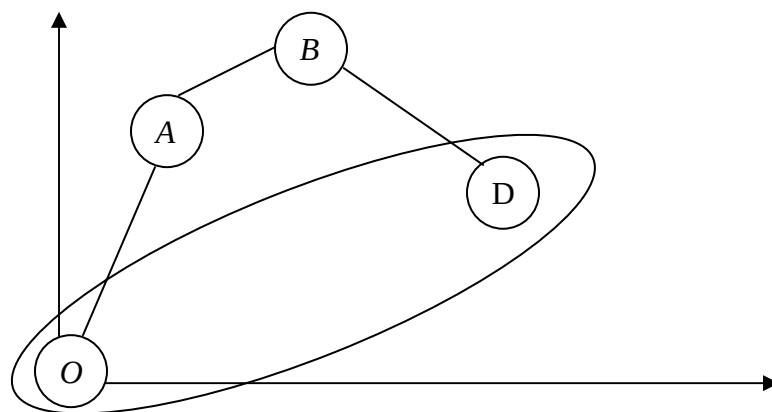


Figure 3.8-5: Ellipsoidal Region Technique precluding any path from O to D .

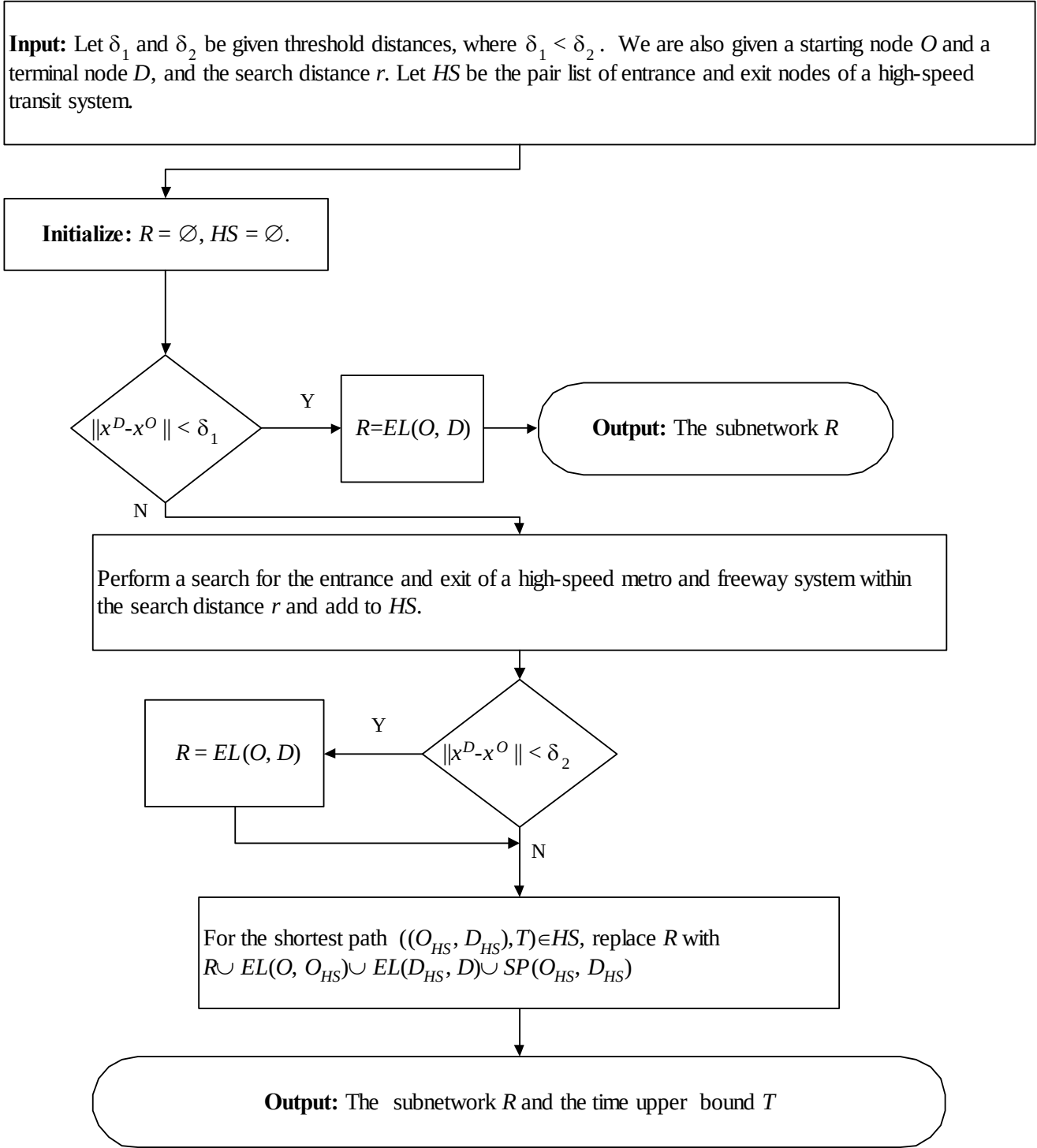


Figure 3.8-6: Flow-chart of the Distance-Based A-ERT rules to reduce the subnetwork to be used in ADTDL CSP.

3.9 Computational Results

We test here four algorithms for solving the approach-dependent, time-dependent label-constrained shortest path problem. These algorithms are:

- (1) Partitioned Shortest Path (PSP) algorithm (Figure 3.6-1);
- (2) Heap-Dijkstra algorithm (Figure 3.6-3);
- (3) Early Terminated PSP algorithm (ET-PSP);
- (4) Early Terminated Heap-Dijkstra algorithm (ETHP-Dijkstra).

We also apply and compare the three heuristic methods; ERT, A-ERT, and the Distance-Based A-ERT. Because of the redundancy of the search process in A-ERT, this causes it to consume too much computational effort. Hence, we modified this method to simultaneously search for freeway sections and paths that lie only within the ellipsoidal regions C_1, C_2 , or E_1 as shown in Figure 3.8-3. If any node lies outside these regions, the algorithm includes this node into the search process only if the link leading to it supports a speed greater than or equal to 20 m/sec. In addition, we also implemented an additional ellipsoidal region technique to curtail the search for the freeway system by combining this with the Distance-Based rules proposed in Section 3.8. We name this approach as the *Distance-Based Augmented Ellipsoidal Region Technique-2* (Distance-Based A-ERT2). The eccentricity of the small ellipse (ratio of minor to major axes lengths) was taken as 0.75 and that of the large ellipse was taken as 0.90. We also combined the Ellipsoidal Region Technique with ET-PSP and ETHP-Dijkstra, and call them *ERT-ET-PSP* and *ERT-ETHP-Dijkstra*, respectively. All of these are compared with the original Route Planner Module (ROUTER), which uses a Heap-Dijkstra algorithm, and is

implemented in the Linux operating system, using C++ along with the g++ GNU compiler.

The tests were performed on the Blacksburg-light, Blacksburg-full, and BigNet networks. The Blacksburg-light network contains only the major streets in Blacksburg, Virginia, and consists of 36 bidirectional links, 31 unidirectional links, and 52 nodes, whereas the Blacksburg-full network contains all the streets encompassing 1251 bidirectional links, 136 unidirectional links, and 1135 nodes. The BigNet network is a transportation network system of Portland, Oregon, containing 7441 bidirectional links, and 3853 nodes. Figures 3.9-1, 3.9-2 and 3.9-3 respectively depict the Blacksburg-light, Blacksburg-full, and BigNet networks.

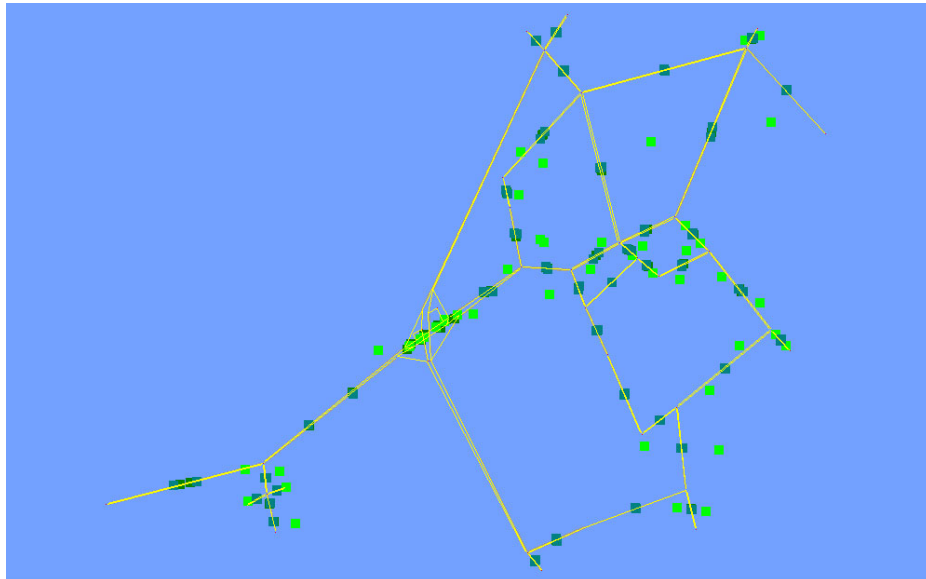


Figure 3.9-1: Blacksburg-light network consists of 36 bidirectional links, 31 unidirectional links, and 52 nodes.

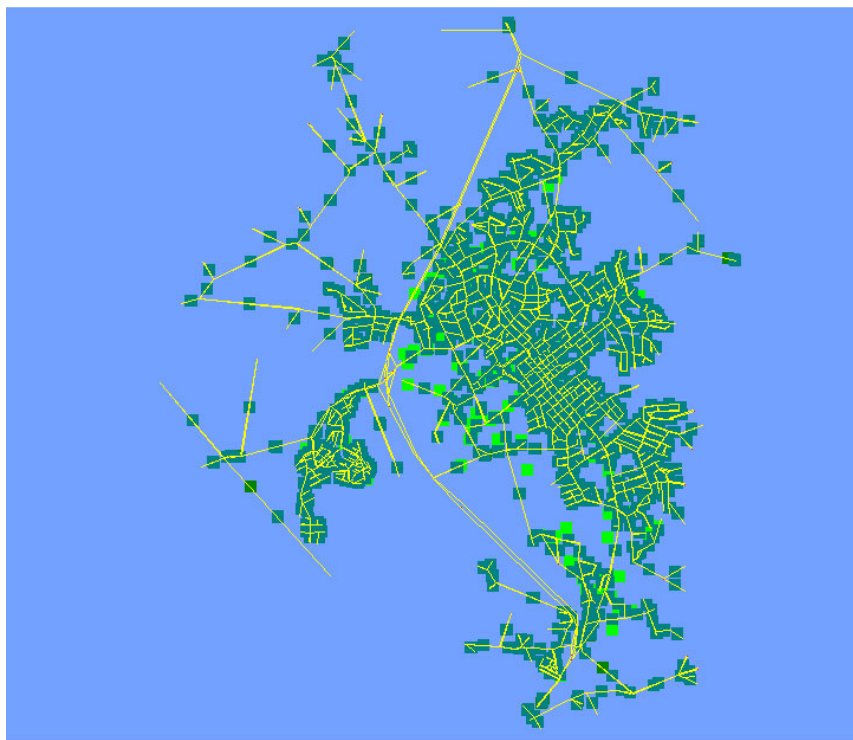


Figure 3.9-2: Blacksburg-full network consists of 1251 bidirectional links, 136 unidirectional links, and 1135 nodes.

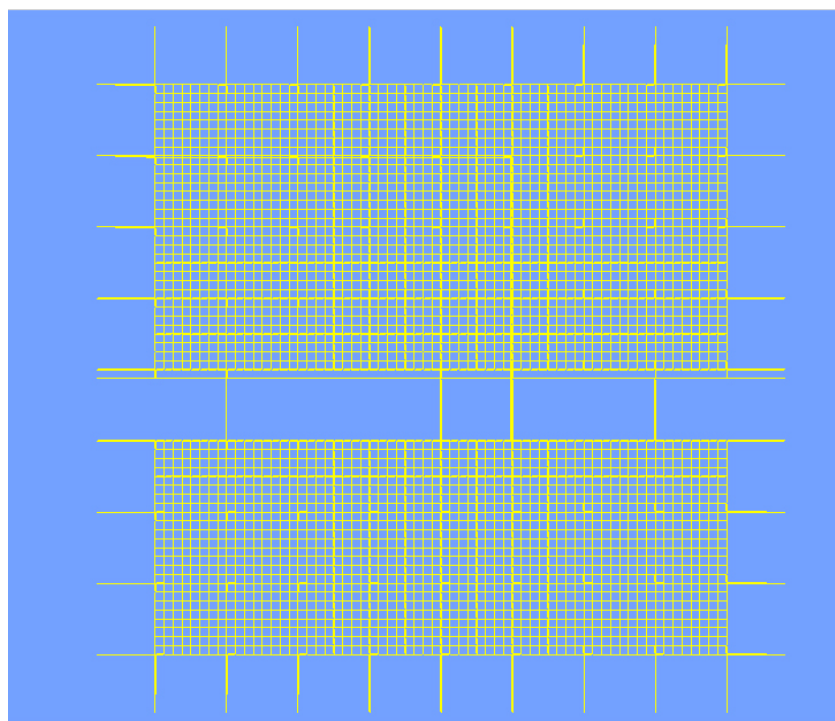


Figure 3.9-3: BigNet network consists of 7441 bidirectional links, and 3853 nodes.

Our coded algorithms accept as input the Approach-Dependent Link Travel Times (ADLTT) summary data. The ADLTT summary data contains the approach node and link for each original node and link. The format of the ADLTT summary data is shown in Table 3.9-1.

Field	Interpretation
LINK	The link ID being reported.
NODE	The node ID from which the vehicles are traveling away.
TIME	The current time (seconds from midnight).
COUNT	The number of vehicles leaving the link.
SUM	The sum of the vehicle travel times (in seconds) for vehicles leaving the link. (The time spent at the previous intersection is included in this value.)
SUMSQUARES	The sum of the vehicle travel time squares (in seconds squared) for vehicles leaving the link. (The time spent at the previous intersection is included in this value.)
TURN	The type of turn the vehicle made when leaving the link: 0 = straight direction (no turn) 1 = right turn -1 = left turn 2 = hard right turn -2 = hard left turn values 3 to 6 represent increasingly more extreme right turns values -3 to -6 represent increasingly more extreme left turns -7 = reverse direction (U-turn).
LANE	The lane number.
VCOUNT	The number of vehicles on the link.
VSUM	The sum of vehicle velocities (in meters per second) on the link.
VSUMSQUARES	The sum of the squares of the vehicle velocities (in meters squared per second squared).
PREVNODE	The previous node ID from which the vehicles are traveling away.
PREVLINK	The previous link ID from which the vehicles are traveling away.

Table 3.9-1: Approach-Dependent Link Travel Times (ADLTT) summary data field records.

We take the original Link Travel Times summary data, which is the output from the Microsimulator, and convert this into the ADLTT summary data using the Lane Connectivity Table and the Turn Prohibition Table. The Lane Connectivity Table consists of information that delineates which link is connected to which other links

through a specified node. The Turn Prohibition Table gives information regarding the specific period of time during which a traveler cannot make a turn from any given link onto another link at a specific node.

To construct the ADLTT summary data, consider the network shown in Figure 3.9-4. We are interested in calculating the link travel time on link (2,3) that is approached via node 1, say, at a specific time period. We first average the travel times for users on link (2,3) who continue through link (3,5), and we call this a *through movement travel time* of link (2,3). Then we average the travel times for users who traverse link (1,2) and make a left turn onto link (2,3), and we call this the *turn movement travel time* of link (1,2). We find the *excess turn travel time* of link (1,2) by subtracting the through movement travel time of link (1,2) from the turn movement travel time of link (1,2). If the excess turn travel time of link (1,2) is positive, we can find the travel time on link (2,3) with the approach from link (1,2) for that specific time period by adding the through movement travel time of link (2,3) and the excess turn travel time of link (1,2). Note that if there is no through movement travel time of link (1,2), we simply take the excess turn travel time of link (1,2) to be zero.

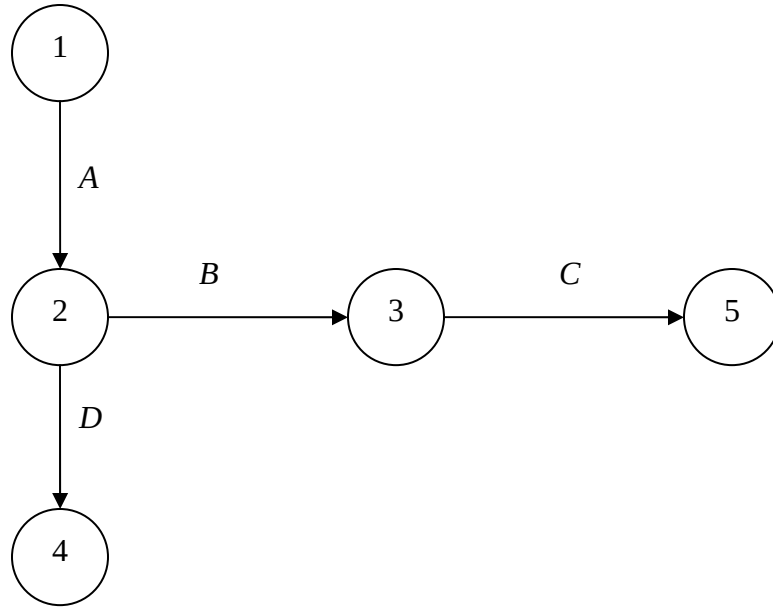


Figure 3.9-4: Example network to calculate the ADLTT summary data.

The JAVA program that we used to convert the link delay travel time summary data is called ApproachFormatter. We can also use this program to generate the ADLTT summary data without using the Link Travel Time summary data, which is an output from the Microsimulator. Without such information, the program will use free-flow speeds to calculate the link travel times. A flow-chart of the interactions between the ApproachFormatter, our coded algorithms or the New Router, and the Microsimulator is depicted in Figure 3.9-5.

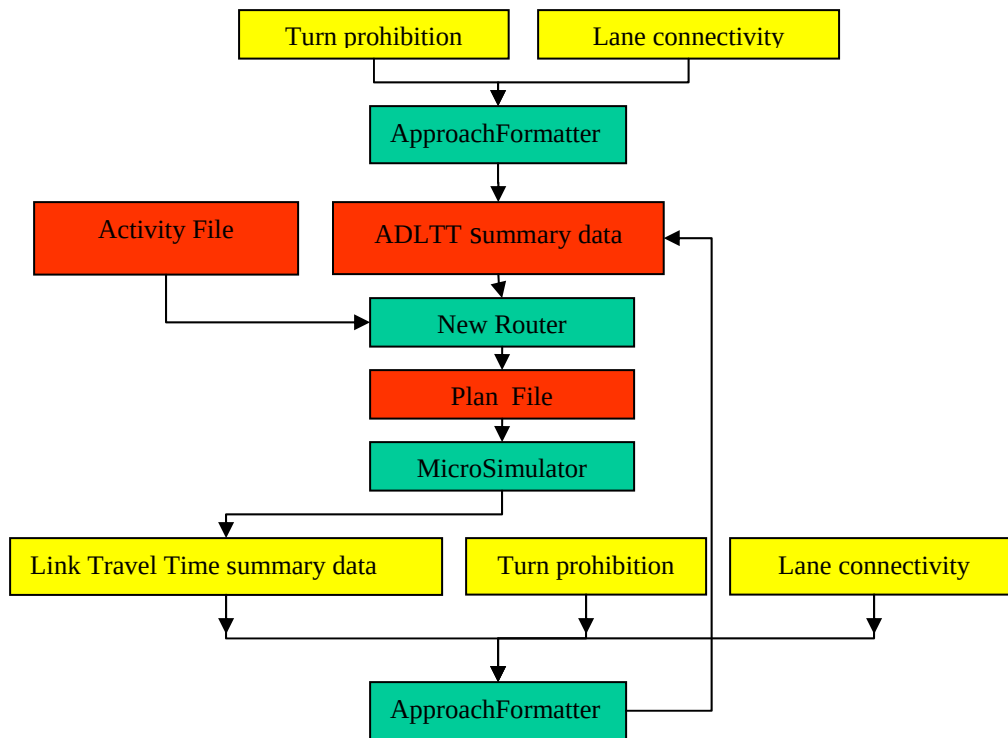


Figure 3.9-5: Flow-chart of the interactions between the ApproachFormatter, the New Router, and the Microsimulator.

At the top of the flow-chart, the ApproachFormatter takes two input tables to generate the ADLTT summary data, which the New Router takes as an input. The New Router solves the ADTDL CSP problem for each requested trip in the Activity File with respect to the travel time specified in the ADLTT summary data. The plan file, which is the result from the New Router, is in the same format as used in the Route Planner Module of TRANSIMS (for further details, see [Los Alamos National Laboratory (2002a)]). The TRANSIMS Microsimulator module simulates the movement and the interactions of users in the transportation system of the study area. In this module, every user tries to execute the travel movements according to the plan file. These movements and interactions produce the Link Travel Time summary data that represents the true traffic link travel times. We use the Link Travel Time summary data along with the

Lane Connectivity Table and the Turn Prohibition Table to generate the ADLTT summary data. Accordingly, we can use the new ADLTT summary data as an input to the New Router to solve the ADTDL CSP problem, which may produce a different path for each user. Each of the paths in the plan file given by the New Router is optimal with respect to the ADLTT summary data.

Table 3.9-2 presents a comparison of the performance of the Heap-Dijkstra algorithm and the PSP algorithm with that for the original Route Planner Module or ROUTER. This experiment was conducted on a Dell 400SC 2.8 GHz computer having 1 GB of memory. The results indicate that the Heap-Dijkstra algorithm has only a marginally higher running time than the original ROUTER. However, the PSP algorithm has a much higher running time than that for the ROUTER and the Heap-Dijkstra algorithms. The paths given by these three algorithms are exactly the same.

Test Network	Number of activities	Number of plans generated	ROUTER Time (secs)	Heap-Dijkstra algorithm		PSP algorithm	
				Time (secs)	Ratio of Heap-Dijkstra to ROUTER run times	Time (secs)	Ratio of PSP to ROUTER run times
Blacksburg-light	4000	10000	5.89417	6.54901	1.1111	7.617	1.292294
Blacksburg-full	5774	14069	42.3153	44.5359	1.052477	132.769	3.137612
BigNet	12000	30000	374.133	419.266	1.120634	817.205	2.184263

Table 3.9-2: Preliminary performance comparison table of the Heap-Dijkstra algorithm, the PSP algorithm, and the ROUTER.

We next conducted an experiment using three scenarios to demonstrate that the resulting path could change with the specified turn restrictions. The first scenario uses the free-flow speeds, the second scenario uses the generated ADLTT summary data

computed without incorporating the output from the Microsimulator, and the last scenario uses the generated ADLTT summary data without the output from Microsimulator, but while considering the turn prohibition at each specific node. This experiment was performed using the Blacksburg-light network for the origin-destination pair identified in Figure 3.9-6. We prohibited the left turn at node 9 when approaching this location from node 902 in the last scenario.

The shortest paths given by the Heap-Dijkstra algorithm for each scenario are as follows:

Scenario 1: 902, 9, 8

Scenario 2: 902, 9, 8

Scenario 3: 902, 801, 8.

The shortest paths for the first two scenarios are the same because the Heap-Dijkstra algorithm uses the free-flow speeds. However, in the last scenario, because we prohibit the left turn at node 9 when approaching this node from node 902, the shortest path for the same origin-destination changes to another path.

We then compared the ROUTER, the PSP algorithm, the Heap-Dijkstra algorithm, and the different proposed heuristics. For this experiment, we used $v = 20$ m/sec in Equation (18) based on the average speed for cars and $\alpha_{i1} = 1, \alpha_{i2} = 2$, and $\alpha_{i3} = 3$. In addition, the upper bound T on the acceptable total travel time was obtained for each trip from the output of the Activity Generator Module of TRANSIMS. For the Blacksburg-light network, 64,800 activities were generated from the Activity Generator Module, and for the Blacksburg-full and BigNet networks, 47,951 activities and 51,200 activities were respectively generated.

We compare below fifteen algorithms, which are numbered as follows:

1. ROUTER
2. Heap-Dijkstra algorithm
3. ETHP-Dijkstra algorithm
4. Ellipsoidal Region Technique Heap-Dijkstra algorithm (ERTHP-Dijkstra algorithm)
5. ERT-ETHP-Dijkstra algorithm
6. Augmented Ellipsoidal Region Technique Heap-Dijkstra algorithm (A-ERTHP-Dijkstra algorithm)
7. Distance-Based A-ERTHP-Dijkstra algorithm
8. Distance-Based A-ERT2HP-Dijkstra algorithm
9. PSP algorithm
10. ET-PSP algorithm
11. Ellipsoidal Region Technique PSP algorithm (ERT-PSP algorithm)
12. ERT-ET-PSP algorithm
13. Augmented Ellipsoidal Region Technique PSP algorithm (A-ERT-PSP algorithm)
14. Distance-Based A-ERT-PSP algorithm
15. Distance-Based A-ERT2-PSP algorithm

Test Network	1	2	3	4	5	6	7	8
Blacksburg-light	78.7575	87.2851	77.4228	73.8014	73.0521	81.9891	86.4908	76.3574
Blacksburg-full	368.632	396.107	260.55	333.154	249.205	366.681	346.943	343.062
BigNet	1867.25	2209.66	911.072	1588.39	902.758	1881.92	1474.97	1458.1

Table 3.9-3: Comparison of *CPU* Times (secs).

Test Network	9	10	11	12	13	14	15
Blacksburg-light	98.4204	89.11	81.7666	78.2194	95.0862	82.513	83.2179
Blacksburg-full	1392.95	1085.63	992.455	937.682	1069.15	1060.01	1058.29
BigNet	4559.7	4017.56	2732.47	2948.09	3264.67	2525.04	2415.04

Table 3.9-4: Comparison of *CPU* Times (secs) (continued).

Test Network	1	2	3	4	5	6	7	8
Blacksburg-light	1	1	1.28068642	1.005421397	1.157461306	1.016144566	1.005421397	1.005421397
Blacksburg-full	1	1	1.012750767	1.000329892	1.011736628	1.000907546	1.000328135	1.000328135
BigNet	1	1	1.129541693	1.000120166	1.122947296	1	1.000010805	1.000010805

Table 3.9-5: Comparison of the *Average Solution Quality*.

Test Network	9	10	11	12	13	14	15
Blacksburg-light	1	1.302990123	1.005421397	1.174697646	1.016144566	1.005421397	1.005421397
Blacksburg-full	1	1.122081905	1.000329453	1.123051149	1.00090711	1.000327696	1.000328135
BigNet	1	1.138238493	1.000120166	1.131812976	1	1.000010805	1.000010805

Table 3.9-6: Comparison of the *Average Solution Quality* (continued).

Test Network	1	2	3	4	5	6	7	8
Blacksburg-light	100	100	76.5462963	86.00925926	66.43518519	90.31481481	86.00925926	86.00925926
Blacksburg-full	100	100	82.55464885	95.4758784	79.89514185	96.12278551	95.52238806	95.52238806
BigNet	100	100	36.25472888	97.43852459	36.10892182	99.98029634	98.4434111	98.4434111

Table 3.9-7: Comparison of the % of *Runs that Yielded Optimal Solutions*.

Test Network	9	10	11	12	13	14	15
Blacksburg-light	100	65.04320988	86.00925926	57.38580247	90.31481481	86.00925926	86.00925926
Blacksburg-full	100	14.09665553	95.4758784	12.61680267	96.12278551	95.52238806	95.52238806
BigNet	100	27.94372636	97.43852459	27.82944515	99.98029634	98.4434111	98.4434111

Table 3.9-8: Comparison of the % of *Runs that Yielded Optimal Solutions* (continued).

Test Network		1	2	3	4	5	6	7	8
Blacksburg-light	Number of plans	162000	162000	162000	147256	143528	153756	147256	147256
	Number of transportation plans	32400	32400	32400	28714	27782	30339	28714	28714
Blacksburg-full	Number of plans	118904	118904	112988	115360	109776	116108	115360	115360
	Number of transportation plans	23651	23651	22172	22765	21369	22952	22765	22765
BigNet	Number of plans	127776	127776	127776	125468	125468	127756	126224	126224
	Number of transportation plans	25376	25376	25376	24799	24799	25371	24988	24988

Table 3.9-9: Comparison of the number of plans and the number of transportation plans found by each algorithm.

Test Network		9	10	11	12	13	14	15
Blacksburg-light	Number of plans	162000	162000	147256	143528	153756	147256	147256
	Number of transportation plans	32400	32400	28714	27782	30339	28714	28714
Blacksburg-full	Number of plans	118904	112988	115360	109776	116108	115360	115360
	Number of transportation plans	23651	22172	22765	21369	22952	22765	22765
BigNet	Number of plans	127776	127776	125468	125468	127756	126224	126224
	Number of transportation plans	25376	25376	24799	24799	25371	24988	24988

Table 3.9-10: Comparison of the number of plans and the number of transportation plans found by each algorithm (continued).

Tables 3.9-3 to 3.9-8 present the results for the CPU time, the Average Solution Quality (ASQ) given by the travel time for the prescribed (heuristic) path divided by the optimal travel time solution value, and the percentage of runs that attained an optimal solution. Note here that our results for each traveler are the information about the transportation activities, which comprise the route path (nodes, links, and travel modes), and the total travel time. The travel modes can be walk, bus, or car, for example. In addition, the results also consist of the non-transportation activities that start and end at the same location. These tables display the comparison results of ASQ and CPU time for only the transportation plans with the travel mode of car, because the non-transportation plans and the walk plans always yield optimal solutions. The CPU time is the total time that each algorithm takes to solve the ADTDL CSP problems for all the users. Tables 3.9-9 and 3.9-10 present the number of plans and the number of transportation plans found by each algorithm.

The results reveal that both the ETHP-Dijkstra algorithm and the ET-PSP algorithm yield a comparable quality of solutions based on the ASQ values (1.141 and 1.188 at average, respectively). However, the ETHP-Dijkstra algorithm consumes the least CPU time (consuming 416.35 seconds at an average), while the ET-PSP algorithm

does not perform much differently from the PSP algorithm (consuming 1730.77 seconds at an average). In addition, the percentage of runs that found optimal solutions using the ETHP-Dijkstra algorithm is much better than that using the ET-PSP algorithm (65.12% and 35.69% at an average, respectively). The ETHP-Dijkstra algorithm outperforms the ET-PSP algorithm because of the nature of the search. In each step, the ETHP-Dijkstra algorithm picks the minimum key node or state and scans its neighbors. The delay-estimation function penalizes those nodes that are further away from the destination node. Therefore, these nodes end up at the bottom of the heap. Thus, the ETHP-Dijkstra algorithm automatically selects nodes that move the traveler closer to the destination. However, the ET-PSP algorithm scans nodes or states by their level, examining all nodes, even those that do not lead the traveler closer to the destination. The delay estimation function does not help to exclude the nodes until they violate the upper bound T , which may occur at the same time as reaching the destination.

To verify this argument, we implemented a variant of the ETHP-Dijkstra algorithm and the ET-PSP algorithm in which we did not incorporate any delay estimation function. Tables 3.9-11 to 3.9-16 present the results obtained. The results for the ET-PSP algorithm are very similar. However, the advantageous effects of curtailing the search, which enhanced the computational efficiency of the ETHP-Dijkstra algorithm, are now virtually erased. Therefore, the delay estimation function does not improve the performance of the PSP algorithm, but it significantly affects that of the ETHP-Dijkstra algorithm.

Test Network	3	3 with $p=0.05$	3 with $p=0.10$	3 with $p=0.15$	3 with $p=0.20$	3 without the delay estimation function
Blacksburg-light	77.3871	77.5785	81.0275	78.303	81.4654	83.6923
Blacksburg-full	262.578	253.008	255.909	257.557	255.813	401.053
BigNet	910.55	835.94	836.374	834.641	835.619	2143.19

Table 3.9-11: Comparison of *CPU Times* (secs) for the EHP-Dijkstra algorithm and the ET-PSP algorithm without the delay estimation function.

Test Network	10	10 with $p=0.05$	10 with $p=0.10$	10 with $p=0.15$	10 with $p=0.20$	10 without the delay estimation function
Blacksburg-light	91.9861	88.4592	90.0632	100.06	88.3916	85.9239
Blacksburg-full	1084.54	1028.63	1023.07	1023.19	1017.39	932.878
BigNet	4017.56	3938.62	3940.6	3920.65	3922.08	3580.64

Table 3.9-12: Comparison of *CPU Times* (secs) for the EHP-Dijkstra algorithm and the ET-PSP algorithm without the delay estimation function (continued).

Test Network	3	3 with $p=0.05$	3 with $p=0.10$	3 with $p=0.15$	3 with $p=0.20$	3 without the delay estimation function
Blacksburg-light	1.2806864	1.0037762	1.0037762	1.0037762	1.0037762	1.0000022
Blacksburg-full	1.0127508	1.0005078	1.0005078	1.0005078	1.0005078	1.0000922
BigNet	1.1295417	1.0191161	1.0191161	1.0191161	1.0191161	1

Table 3.9-13: Comparison of the *Average Solution Quality* for the EHP-Dijkstra algorithm and the ET-PSP algorithm without the delay estimation function.

Test Network	10	10 with $p=0.05$	10 with $p=0.10$	10 with $p=0.15$	10 with $p=0.20$	10 without the delay estimation function
Blacksburg-light	1.3029901	1.0588173	1.0588173	1.0588173	1.0588173	1.0593846
Blacksburg-full	1.1220819	1.1186812	1.1186812	1.1186812	1.1186812	1.118699
BigNet	1.1382385	1.0540278	1.0540278	1.0540278	1.0540278	1.0540278

Table 3.9-14: Comparison of the *Average Solution Quality* for the EHP-Dijkstra algorithm and the ET-PSP algorithm without the delay estimation function (continued).

Test Network	3	3 with $p=0.05$	3 with $p=0.10$	3 with $p=0.15$	3 with $p=0.20$	3 without the delay estimation function
Blacksburg-light	76.546296	94.574074	94.574074	94.574074	94.574074	99.969136
Blacksburg-full	82.554649	98.291827	98.291827	98.291827	98.291827	99.835102
BigNet	36.254729	89.738335	89.738335	89.738335	89.738335	100

Table 3.9-15: Comparison of the % of *Runs that Yielded Optimal Solutions* for the EHP-Dijkstra algorithm and the ET-PSP algorithm without the delay estimation function.

Test Network	10	10 with $p=0.05$	10 with $p=0.10$	10 with $p=0.15$	10 with $p=0.20$	10 without the delay estimation function
Blacksburg-light	65.04321	68.87963	68.87963	68.87963	68.87963	68.333333
Blacksburg-full	14.096656	23.026511	23.026511	23.026511	23.026511	23.026511
BigNet	27.943726	60.872478	60.872478	60.872478	60.872478	60.872478

Table 3.9-16: Comparison of the % of *Runs that Yielded Optimal Solutions* for the ETHP-Dijkstra algorithm and the ET-PSP algorithm without the delay estimation function (continued).

The results from tables 3.9-5 and 3.9-6 show that both the ERTHP-Dijkstra algorithm and the ERT-PSP algorithm yield the same ASQ values and the percentage of runs that result in finding optimal solutions for the Blacksburg-light and BigNet networks. This phenomenon is also evident when comparing the A-ERTHP-Dijkstra, Distance-Based A-ERTHP-Dijkstra, and Distance-Based A-ERT2HP-Dijkstra algorithms; as well as in the comparison of the A-ERT-PSP, Distance-Based A-ERT-PSP, and Distance-Based A-ERT2-PSP algorithms. However, the results from tables 3.9-3 and 3.9-4 show that all of the variants of Heap-Dijkstra algorithm perform faster than those of the PSP algorithm. Therefore, we will henceforth focus on the variants of the Heap-Dijkstra algorithm.

The ETHP-Dijkstra and the ERT-ETHP-Dijkstra algorithms perform faster than any of the variants of Dijkstra algorithm (consuming 416.35 and 665.12 seconds at an average, respectively), but the ASQ values (1.14099296 and 1.097381743 at an average, respectively) and the percentage of runs that result in finding optimal solutions (65.11855801% and 60.81308295% at an average, respectively) appear to be worse. The A-ERTHP-Dijkstra algorithm performs marginally slower than the ERTHP-Dijkstra algorithm (776.8633667 and 665.1151333 seconds at an average, respectively), but the quality of solutions it finds are the best (95.47% at an average). Both the Distance-Based A-ERTHP-Dijkstra and Distance-Based A-ERT2HP-Dijkstra algorithms provide the

same quality solutions (1.001920113 at an average), which are better than those produced by the ERTHP-Dijkstra algorithm (1.001957152 at an average). However, the Distance-Based A-ERT2HP-Dijkstra algorithm consumes less CPU time than the Distance-Based A-ERTHP-Dijkstra algorithm for the entire test networks (625.8398 and 636.1346 seconds at an average, respectively). The Distance-Based A-ERT2HP-Dijkstra algorithm and the ERTHP-Dijkstra algorithm consume a comparable CPU time on the Blacksburg-light and Blacksburg-full networks (209.7097 and 203.4777 seconds at an average, respectively), but the Distance-Based A-ERT2HP-Dijkstra algorithm saves about 34% of the CPU effort while the ERTHP-Dijkstra algorithm saves only about 28.1% of the CPU effort for the BigNet network.

From tables 3.9-13 to 3.9-16, we see that the EHP-Dijkstra algorithm, for different values p ranging from 0.05 to 0.20, performs about the same with respect to both the ASQ values and the percentage of runs that result in finding optimal solutions (1.0078 and 94.20% at an average, respectively). The same holds true for the ET-PSP algorithm (1.077175 and 50.92% at an average, respectively). However, the EHP-Dijkstra with $p=0.05$ performs faster than the original EHP-Dijkstra (consuming 388.8422 versus 416.8384 seconds at an average, respectively), while it improves both the ASQ values and the percentage of runs that result in finding optimal solutions. It is noted here that the number of plans and the number of transportation plans found by both the EHP-Dijkstra and ET-PSP methods are precisely the same as those found by the exact method.

The Distance-Based A-ERT2HP-Dijkstra algorithm performs marginally better than the EHP-Dijkstra with $p=0.05$ in terms of the ASQ values (1.0019 versus 1.0078 at

an average, respectively). However, the Distance-Based A-ERT2HP-Dijkstra algorithm produced only 94.449% of transportation plans, while the ETHP-Dijkstra with $p=0.05$ produced all of them. In addition, the percentage of runs that result in finding optimal solutions for the ETHP-Dijkstra algorithm with $p=0.05$ is better than that of the Distance-Based A-ERT2HP-Dijkstra algorithm (94.20% versus 93.33% at an average, respectively). Moreover, the ETHP-Dijkstra with $p=0.05$ saves CPU effort by 56.68% at an average, which is far better than the savings of 30.28% (at an average), as produced by the Distance-Based A-ERT2HP-Dijkstra algorithm.

3.10 Summary and Conclusions and Recommendation for Future Research

We have proposed two solution methods, the Heap-Dijkstra (HP-Dijkstra) algorithm and the Partitioned Shortest Path (PSP) algorithm, for the approach-dependent, time-dependent, label-constrained shortest path problem. This class of problems is an extension of the time-dependent label-constrained shortest path problem that was discussed by Sherali *et al.* (2003), and is used to model turn-penalties in transportation networks. We model this problem by incorporating within each link's travel time function a dependence on the link via which its tail node was approached. This model also allows us to include turn prohibition(s) during specific period(s) of time when a traveler cannot make a turn from a given link onto another link at a particular node. We adapt the method discussed by Sherali *et al.* (2003) to solve this problem by implicitly generating a composite network G^* based on information from G and G_L . Furthermore, we study two heuristic schemes discussed by Sherali *et al.* (2003) to curtail the search by

growing the search tree more pointedly in the direction from the origin to the destination. In addition, we have proposed four additional heuristic methods to include high speed corridors into the search. The proposed exact and heuristic methods have been implemented in the Linux operating system, using C++ along with the g++ GNU compiler. We have tested twenty-five different combinations of network curtailment and algorithmic strategies on three test networks: the Blacksburg-light, the Blacksburg-full, and the BigNet network. The Blacksburg-light network contains only the major streets in Blacksburg, Virginia, and consists of 36 bidirectional links, 31 unidirectional links, and 52 nodes, whereas the Blacksburg-full network contains all the streets encompassing 1251 bidirectional links, 136 unidirectional links, and 1135 nodes. The BigNet network is a transportation network system of Portland, Oregon, containing 7441 bidirectional links, and 3853 nodes. For the Blacksburg-light network, we have 64,800 activities that have been generated by the Activity Generator Module, and for the Blacksburg-full and BigNet networks, we have 47,951 activities and 51,200 activities, respectively. Our computational experiments were conducted on a Dell 400SC 2.8 GHz computer having 1 GB of memory. The results indicate that the Heap-Dijkstra implementations are much faster than the PSP algorithmic approaches for solving the underlying problem exactly. Furthermore, among the curtailment schemes, the EHP-Dijkstra with $p=0.05$, based on estimating a lower bound on the travel time from any intermediate node i to the terminal node D , yields the best overall results. This method produces solutions within 0.37-1.91% of optimality, while decreasing CPU effort by 56.68% at an average, as compared with applying the best available exact algorithm. Our work suggests that it would be beneficial to test these methods on different networks because the results can vary quite

significantly based on the network structure. Also, such a wider investigation might lead additional insights into composing alternative network curtailment strategies.

Chapter 4

ENHANCEMENTS FOR THE CART ALGORITHM WITH APPLICATION TO TRANSPORTATION SYSTEMS

4.1 Introduction and Motivation

The Classification and Regression Tree algorithm (CART) (see [Breiman *et al.* (1984)]) has been extensively used by transportation engineers and planners to conduct analyses of their survey data to provide information regarding the correlation between some independent household demographic X -variables and some dependent activity time or trip time Y -variables for these households. This classification and correlation process permits the user to explore different combinations of dependent and independent variables. However, the user does not have control or direction on how to achieve some desired objectives in trip or activity generation within this CART process, such as the maximization of the number of trips produced for all households, or maximizing the average number of trips per household that exceed a minimum threshold.

The motivation for this research arose from an application of the CART algorithm to the next generation planning/simulation software package TRANSIMS. This application focuses on attempting to use different combinations of X - and Y -variables for each household in the CART implementation to achieve a maximum number of trips for all households [Los Alamos National Laboratory (2002a)]. However, the approach adopted to accomplish this requires a large number of trials of different combinations of

X- and Y-variables, but yet does not guarantee an optimal outcome in any defined sense. It became evident to us that the best way to achieve this would be to modify the CART algorithm itself, particularly in the pruning phase, so that the user could avoid extensive trial and error attempts, and yet quickly attain the desired objectives.

The main contribution of this research is to develop a new concept of formulating suitable optimization models that can conduct pruning of classification trees based on specified objective or merit functions and side-constraints that govern certain structural and statistical properties of the generated classification trees. We also apply this general concept within the context of the Activity Generator Module of TRANSIMS.

The remainder of this chapter is organized as follows. In Section 4.2 below we provide an overview of TRANSIMS and its Activity Generator Module, which is the subject of principal focus in this work, along with some general concepts and enhancements related to classification tree development and pruning strategies. Some discussion is repeated from previous chapters to make this presentation self-contained. Next, in Section 4.3, we present a new idea of formulating optimization models to conduct optimal pruning based on specified objective functions and subject to desired side-constraints. Finally, in Section 4.4, we conclude this chapter with a summary of the proposal research tasks.

4.2 An Overview of TRANSIMS

TRANSIMS (Transportation Analysis and Simulation System) is an integrated package of travel forecasting models designed to provide transportation planners

comprehensive information on traffic impacts, congestion, and pollution [Los Alamos National Laboratory (2002a)]. The TRANSIMS software is developed by the Los Alamos National Laboratory (LANL), and its modules create a virtual metropolitan region with a complete representation of the region's individuals, their activities, and the transportation infrastructure. Trips are planned to satisfy the individuals' activity patterns. TRANSIMS then simulates the movement of individuals across the transportation network, including their use of vehicles such as cars or buses, on a second-by-second basis. This virtual world of travelers mimics the traveling and driving behavior of real people in the region. The interactions of individual vehicles produce realistic traffic dynamics from which analysts using TRANSIMS can estimate vehicle emissions and judge the overall performance of the transportation system.

There are six principal modules within TRANSIMS: The Population Synthesizer, the Activity Generator, the Route Planner, the Microsimulator, the Emissions Estimator, and the Selector module (See [Los Alamos National Laboratory (2002a)] for details regarding these modules and their inter-relationships). Each module can be easily replaced or modified without revamping the entire TRANSIMS framework. Additionally, new modules may be added without much difficulty. Figure 4.2-1 provides an overview of the framework of TRANSIMS. Our focus in this chapter will be on the Activity Generator module.

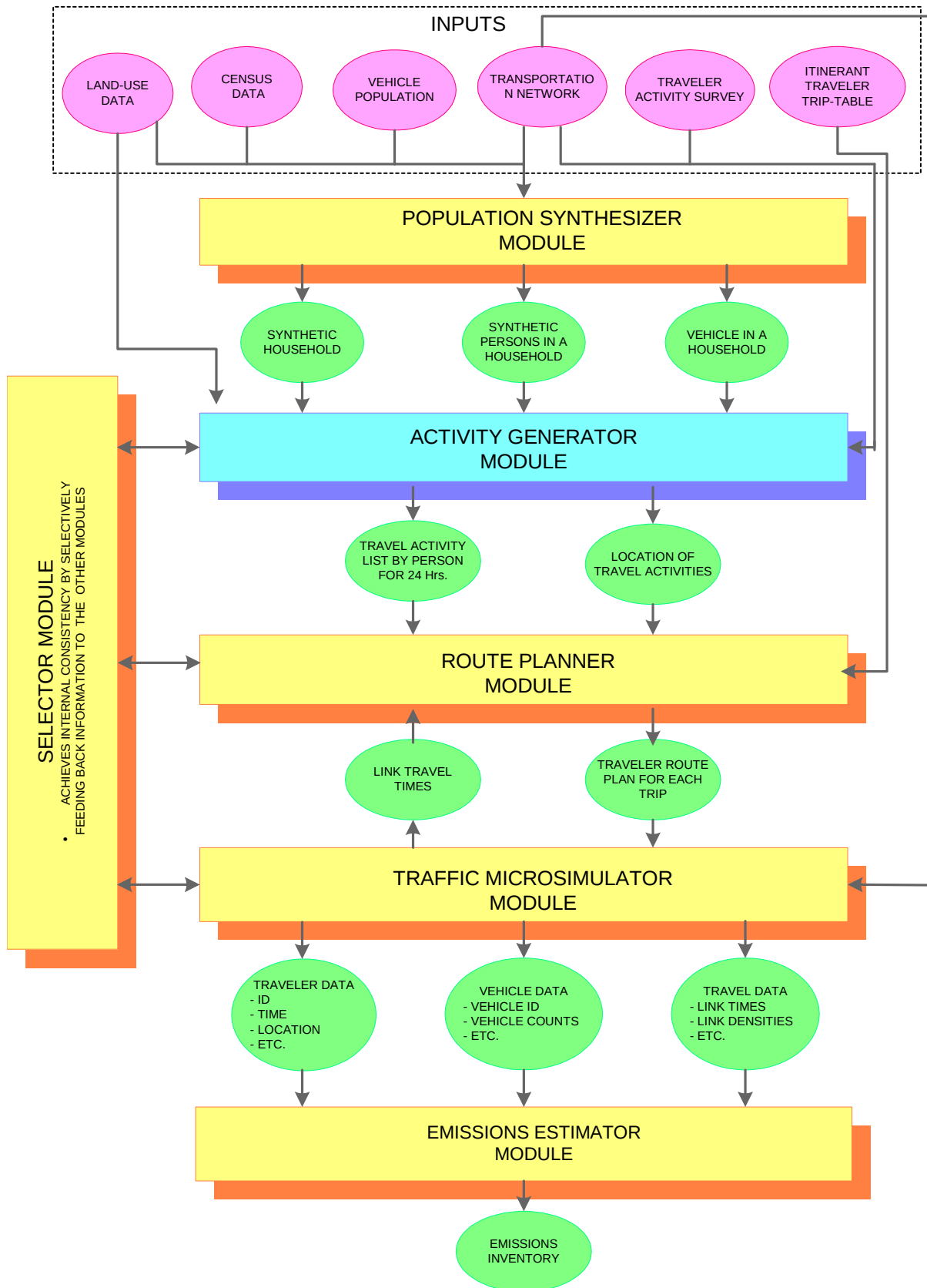


Figure 4.2-1: TRANSIMS framework.

4.2.1 Activity Generator in TRANSIMS

The Activity Generator module in the TRANSIMS framework develops a list of activities for each member of a synthetic household over a 24-hour horizon. This module requires as principal inputs the synthetic households (obtained from the Population Synthesizer module), the study area activity survey of households, itinerant travel data, the TRANSIMS network data, and land-use data. Using these inputs, the Activity Generator module produces a list of activities for each traveler [Los Alamos National Laboratory (2002a)].

The synthetic households are generated by the Population Synthesizer module using census data that represent the real households in the metropolitan area under study. The demographic data of the synthetic households must match the demographic data of the survey households that are part of the Activity Generator's database. These demographic data are used to select a suitable survey household to match any given synthetic household. This matching is then used to produce the activity patterns for the members of the synthetic household.

The survey households are comprised of a set of households that have been surveyed to yield the input database for the Activity Generator module. The data from the survey household consist of information related to household demographic data and household activity data. The survey household demographic data contains information about the characteristics of each household member in the survey sample of households. This data is used for building and matching the demographic characteristics of the synthetic households. The activity survey obtained from the survey household is a

representative sample of the household activities, including travel and event-participation information for each household member. Skeletal activity patterns are created by stripping locations from the database.

The Activity Generator module uses household demographic survey data to build a classification tree by applying the CART (Classification and Regression Tree) algorithm. Each survey household is effectively placed by the CART algorithm into one of the tree's terminal nodes. The survey household demographic data pertaining to each of the terminal nodes of this tree is used to match a given synthetic household with a survey household within the particular terminal node of the tree. The format of the synthetic household demographic file is the same as the survey household demographic file.

The Activity Generator in TRANSIMS uses a model that considers the time spent for various activities as dependent variables, which provide a basis for comparison to match the synthetic households to the survey households. The assumption made in this model is that the activity travel patterns of households would be similar if the time spent for these activities is also similar.

4.2.2 Algorithms in the Activity Generator

As discussed above, the role of the Activity Generator in TRANSIMS is to develop a list of activities for each member of a synthetic household over a 24-hour horizon. The algorithms used to generate the list of activities for each member of a synthetic household involve the following five steps:

1. Create skeletal activity patterns from the survey households.
2. Use the CART (Classification and Regression Tree) algorithm to build a classification tree based on the household demographic data.
3. Match the given synthetic household with a survey household.
4. Generate activity times and durations.
5. Generate destination locations.

The focus of this research effort will be on the first three steps, and in particular, on the CART algorithm, used to produce an accurate classification of household characteristics based on their travel behavior [Los Alamos National Laboratory (2002a)]. This algorithm is comprised of two steps: the tree growing step and the cross-validation step for reducing or pruning the tree. In the tree growing step the CART algorithm constructs a binary tree structure by repeatedly splitting each active node into two subnodes based on a split-criterion and a split-value. The two new subnodes become, in turn, active nodes. These splits are performed until all leaf nodes can be declared to be terminal nodes. A brief description of the tree growing algorithm is as follows [Los Alamos National Laboratory (2002a)]:

Consider the following notation.

- L : survey sample comprised of n data observations, indexed $i = 1, \dots, n$.
- $X_k, k = 1, \dots, K$: Given set of K demographic attributes. (These are the *independent* variables.) For example, X_1 = household size, X_2 = number of children with ages $\in [0, 5]$ in the household, X_3 = number of children with ages $\in [5, 17]$ in the household, etc.
- X_{ik} = value of demographic attribute X_k in the i^{th} survey observation, $i = 1, \dots, n, k = 1, \dots, K$.
- $Y_j, j = 1, \dots, p$: Given set of p *dependent* variables used to measure the performance characteristics of the households. In TRANSIMS, there are $p = 16$ such variables. Fifteen of these measure the total (weighted) time spent by the household in each of some 15 broadly classified activity types, and the 16th one pertains to the total number of trips made by the household over the specified 24-hour horizon.
- Y_{ij} = value of measure Y_j for the i^{th} survey observation, $i = 1, \dots, n, j = 1, \dots, p$.
- $s_j = 1/\text{variance}(Y_j)$ (e.g., as estimated from L), $\forall j = 1, \dots, p$.
- $D_j(N) = \text{deviance (or impurity) of a subset (or node) } N \subseteq L \text{ with respect to measure } j$, for $j = 1, \dots, p$. This is computed as the sum of squared deviations of $Y_{ij}, \forall i \in N$, from their mean:

$$D_j(N) = \sum_{i \in N} (Y_{ij} - \bar{Y}_{Nj})^2, \text{ where } \bar{Y}_{Nj} \equiv \sum_{i \in N} Y_{ij} / |N|. \quad (1)$$

$$D(N) = \sum_{j=1}^p s_j D_j(N) = \text{total (weighted) deviance (or impurity) of node } N \subseteq L. \quad (2)$$

- $t_k(N) = \{\text{possible values of } t \text{ such that there exist observations } i \text{ in } N \text{ having } X_{ik} \leq t \text{ and others having } X_{ik} > t\}$, for each $N \subseteq L$, and $k = 1, \dots, K$.

- Partition $\{N_{1kt}, N_{2kt}\}$ of N : For each $k = 1, \dots, K$, and $t \in t_k(N)$, where $N \subseteq L$, define

$$N_{1kt} = \{i \in N : X_{ik} \leq t\} \text{ and } N_{2kt} = \{i \in N : X_{ik} > t\}. \quad (3)$$

- $\Delta_{kt}(N)$: decrement in the total deviance upon partitioning N into $\{N_{1kt}, N_{2kt}\}$ given by

$$\Delta_{kt}(N) = D(N) - [D(N_{1kt}) + D(N_{2kt})]. \quad (4)$$

The *tree-growing algorithm* then proceeds as follows. At each stage of this process, beginning with the *root node* comprised of L , the procedure selects an active *terminal* or *end* node N and considers splitting or partitioning this node into two subnodes. (A current end node is considered *active* if its deviance exceeds a threshold value $\beta \equiv 0.01D(L)$.) The partitioning of N is performed based on that attribute X_k and its associated value $t \in t_k(N)$ that yields the largest value of the decrement $\Delta_{kt}(N)$ in the total deviance. If the number of observations in both the resulting children nodes is at least some minimal value $N_{\min}$ (taken as 10 in TRANSIMS), this partitioning is conducted. Otherwise, the end node N is declared as inactive, and will end up being a terminal node of the final tree T that is constructed.

Remark 1. Note that aside from the foregoing criteria, we will investigate basing the selection and splitting of nodes on other, possibly multiple, considerations. For example,

one might be interested in ensuring that the average number of trips made by the households that are assigned to each node must exceed some minimal threshold. Denoting $j^* \in \{1, \dots, p\}$ as the index that represents the total number of trips made by the household, for any node N , the average number of trips for the assigned households would then be given by

$$F_{j^*}(N) = \frac{\sum_{i \in N} Y_{ij^*}}{|N|}. \quad (5)$$

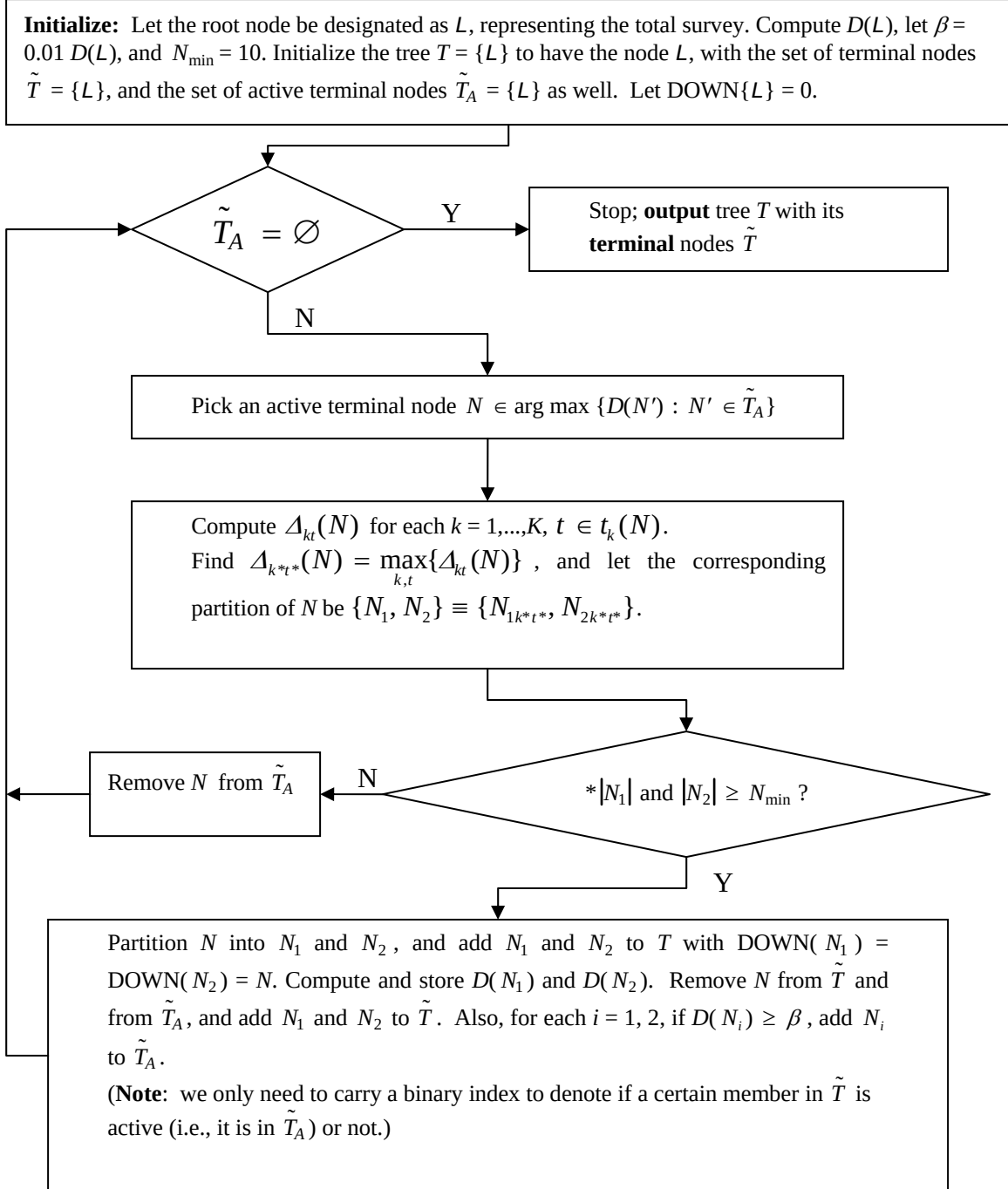
For some threshold $F_{\min}$, we may ensure that $F_{j^*}(N) \geq F_{\min}$ for each end node that is generated by choosing to partition any node N into subnodes N_1 and N_2 , only if

$$|N_i| \geq N_{\min} \text{ and } F_{j^*}(N_i) \geq F_{\min} \text{ for each } i = 1, 2. \quad (6)$$

Figure 4.2.2-1 provides a flow-chart for the above tree-growing procedure, where the following additional notation has been introduced.

$\tilde{T}$: Set of terminal nodes of any given classification tree T .

$\tilde{T}_A$: Set of currently active terminal nodes during the tree-growing process.



*Possibly, also check if $F_{j^*}(N_i) \geq F_{\min}$ for each $i = 1, 2$ in this set of conditions as per Remark 1.

Figure 4.2.2-1: Flow-chart for the Tree-Growing Procedure.

4.2.3 Cross-Validation Scheme of CART to Prune the Binary Tree.

Let T be the tree produced via the tree-growing procedure described in Section 2.2. The CART algorithm recommends two alternative schemes for reducing or *pruning* (or *pruning upward*) the tree T , namely, the *cross-validation* approach and the *independent test sample* approach. TRANSIMS implements a variant (called **S-PLUS**) of the former scheme, which is the one recommended by Breiman *et al.* (1984). In this process, the survey sample L is randomly partitioned into V roughly equal subsets $L_1, \dots, L_V$. (The value $V = 10$ is recommended by TRANSIMS.) Letting $L^{(v)} = L - L_v$ for each $v = 1, \dots, V$, the CART algorithm is used to determine a tree $T^{(v)}$ based on the observations in $L^{(v)}$ alone, for each $v = 1, \dots, V$. Presumably, because of the smaller sample size, each of these trees will tend to be smaller than the tree T obtained using the full sample L . Then, based on certain prediction errors pertaining to classifying all the observations in L into the terminal nodes of each $T^{(v)}$, the tree yielding the least prediction error is selected (further details are provided below). However, when the sample size L is very large, Breiman *et al.* recommend the less computationally intensive independent test sample. Remark 3 given at the end of the present section comments on the specifics of this approach. In Section 4.3, following this discussion, we shall present a new *optimal pruning strategy* based on various stated criteria that include diverse considerations such as deviance, number of terminal nodes, and total number of induced trips.

Now, consider any arbitrary tree $\tau \in \{T^{(v)}, v = 1, \dots, V\}$. As before, let $\tilde{\tau}$ denote the set of *terminal nodes* of τ . For any observation i , let $\tilde{\tau}(i)$ be the terminal node from $\tilde{\tau}$ that this observation i would fall into (this would be the *classification node* of i). Compute a *prediction error* for observation i with respect to the tree τ as

$$e_i(\tau) = \sum_{j=1}^p s_j (Y_{ij} - \bar{Y}_{\tilde{\tau}(i),j})^2 \quad (7)$$

where $\bar{Y}_{\tilde{\tau}(i),j}$ (from Equation (1)) is the average of the values of the measure j for the observations that currently reside in the terminal node $\tilde{\tau}(i)$ of the tree τ . Likewise, for any subset $L_v \subseteq L$, define

$$e_{L_v}(\tau) = \sum_{i \in L_v} e_i(\tau) \quad (8)$$

and for the entire set of observations in L , let

$$e(\tau) \equiv e_L(\tau) = \sum_{v=1}^V e_{L_v}(\tau) = \sum_{i=1}^n \sum_{j=1}^p s_j (Y_{ij} - \bar{Y}_{\tilde{\tau}(i),j})^2. \quad (9)$$

Then, the *cross-validation method S-PLUS* in TRANSIMS selects that tree $T^{(v^*)}$ for implementation such that

$$v^* \in \arg \min \{e(T^{(v)}), v = 1, \dots, V\}. \quad (10)$$

Remark 2. TRANSIMS also recommends a pruning process based on the following definitions.

- $D(\tau)$: deviance measure defined via (1)-(2) for any tree τ as the total deviance over its terminal nodes (denoted $\tilde{\tau}$) given by

$$D(\tau) = \sum_{N \in \tilde{\tau}} D(N). \quad (11)$$

- $D_\alpha(\tau)$: *penalty function* that augments the deviance $D(\tau)$ with a penalty term given by some penalty parameter $\alpha > 0$ multiplied by the number of terminal nodes $|\tilde{\tau}|$ in τ . Hence,

$$D_\alpha(\tau) = D(\tau) + \alpha |\tilde{\tau}|. \quad (12)$$

- $\tau_N \subseteq \tau$: *subtree rooted at node N* of τ given by N and its *descendents* (or *successors*) *within* τ .

Given any tree τ (where the initial τ equals the original tree T) and a node N of τ , where $N \notin \tilde{\tau}$, and given any parameter α , we can compute the penalty function value similar to (12) for the *node* N and for the *subtree* τ_N that is rooted at N as follows

$$D_\alpha(N) = D(N) + \alpha \quad (13a)$$

$$D_\alpha(\tau_N) = D(\tau_N) + \alpha |\tilde{\tau}_N|. \quad (13b)$$

Note that from (13 a, b), we will have

$$D_\alpha(\tau_N) < D_\alpha(N) \text{ so long as } \alpha < \alpha(N) \equiv \frac{D(N) - D(\tau_N)}{|\tilde{\tau}_N| - 1}. \quad (14)$$

In other words, for any $\alpha \geq \alpha(N)$, using the penalty function (12), we would prefer collapsing the subtree τ_N rooted at N into this node N . Suppose that we compute this value $\alpha(N)$ given by (14) for all the nonterminal nodes N of τ , and find

$$N^* \in \arg \min \{\alpha(N) : N \text{ is a node of } \tau, N \notin \tilde{\tau}\}. \quad (15)$$

Given any tree τ (where the initial τ equals the original tree T) a node N^* is determined that provides the *weakest junction* of τ in the sense that among the nodes $N \notin \tilde{\tau}, N^*$ yields the smallest value of the penalty parameter α beyond which it becomes preferable to collapse the subtree τ_{N^*} rooted at the node N^* with respect to the merit function (12).

This establishes an inductive process whereby after finding N^* and collapsing τ_{N^*} into N^* , the process is repeated with the *pruned* tree given by $\tau - \tau_{N^*}$. This produces a sequence of *nested trees*, leading ultimately to the single root node r , say, of the original tree. This sequence is then used as discussed in Remark 3 below. $\square$

Remark 3 (Independent Test Sample Approach). This alternative approach to the cross-validation method is more suited for large sample sizes, but is not currently a part of TRANSIMS. We will investigate this approach as well, using the above pruning concept based on the deviance measures (11) and (12). Here, the survey sample set L can first be partitioned into the sets L_1 and L_2 , where L_1 is suitably sized so that the nested sequence of trees described in Remark 2 can be conveniently determined, and L_2 is the *independent test sample* that can subsequently be used to select from among the foregoing nested sequence. This selection can be based on a suitable error prediction function. Figure 4.2.2-2 presents a flow-chart that describes this process. We will also investigate more efficient alternatives that *simultaneously* determine the nested sequence of pruned trees and evaluate the corresponding prediction errors, continuing this pruning process so long as the prediction error remains nonincreasing. Although monotonicity is

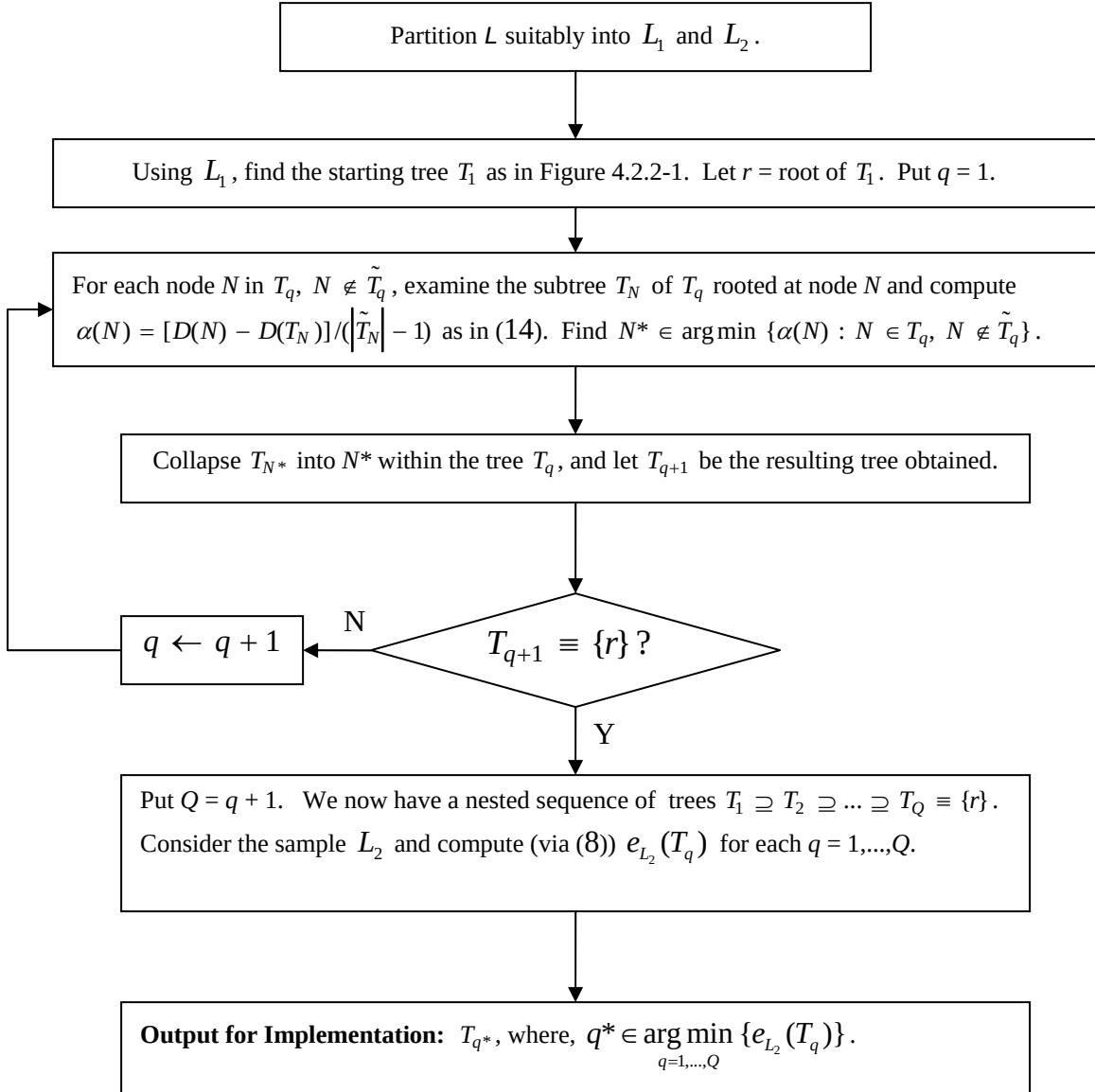


Figure 4.2.2-2: Flow-Chart for the Pruning and the Independent Test Sample Approach.

not guaranteed, at the first increase in the prediction error function, we could terminate this process in order to conserve effort. Moreover, in any case, note that if L_1 is a sufficiently representative sample, and if T_1 is an elaborately expanded classification tree,

then it is quite likely that $q^*=1$ by the procedure of Figure 4.2.2-2, leading to no pruning action. $\square$

4.3 Optimal Pruning Strategy

We now describe in this section a new strategy that can be used to optimally prune a given tree T based on various specified objectives, goals, and side-constraints. To present the proposed methodology, let us begin by introducing some notation and terminology.

As before, let T be the tree obtained via some tree-growing procedure and let $\tilde{T}$ represent its terminal or end nodes. For convenience in notation, we will index the nodes of T as $h = 0, 1, \dots, H$, where node 0 represents the root node, and denote this set of nodes by $\mathcal{H} = \{0, 1, \dots, H\}$. For any $h \in \mathcal{H}$, let us define the following (standard) terminology (see Bazaraa, Jarvis, and Sherali, 1990, for example).

$p(h)$ = set of *predecessor nodes* of $h \equiv$ set of nodes (excluding h) that lie on the chain from h to the root node 0 ($p(0) = \phi$).

$p_i(h) \equiv \text{DOWN}(h) =$ *immediate predecessor* of node $h \equiv$ first node following h that lies on the chain from h to the root node.

T_h = subtree of T that is *rooted at node* h (including h). (Hence, $T_0 \equiv T$.)

$b(h) =$ *brother* of node h (exists for $h \neq 0$) = node for which $p_i[b(h)] = p_i(h)$. Note that $b[b(h)] = h$ since T is a binary tree.)

$s(h)$ = set of *successor nodes* of $h \equiv$ set of nodes in $T_h - \{h\} \equiv$ set of nodes u for which $h \in p(u)$.

$s_I(h)$ = the two *immediate successors* (or *children*) of node h (if they exist) that are produced by partitioning node h as in the tree-growing procedure.

Furthermore, for the sake of convenience, we will henceforth refer to the (unique) chain from any node u to a node v in T , including the nodes u and v , by $C[u \rightarrow v]$. When we wish to exclude nodes u , or v , or both from this chain, we will use the notation $C(u \rightarrow v]$, $C[u \rightarrow v)$, and $C(u \rightarrow v)$, respectively. Finally, the process of pruning a subtree T_h that is rooted at h by absorbing it within the node h will be referred to as *collapsing T_h into h* . Likewise, any node that is a part of such a subtree T_h (excluding the node h) will be called a “*collapsed node*.”

Any legitimate pruning of the tree T will comprise of collapsing entire subtrees rooted at particular nodes into these corresponding nodes, thereby producing a resultant tree $T(E)$ having E as its set of end nodes. A formal definition of a pruned tree is given below.

Definition 1. We will say that $T(E)$ is a *pruned tree of T having end nodes E* iff:

(i) $T(E)$ is a subtree of T that contains the root node 0, has $E \subseteq \mathcal{H}$ as its set of end nodes,

and is given by $\bigcup_{h \in E} C[h \rightarrow 0]$, and

(ii) for any node $h \neq 0$ that is included in $T(E)$, we also have its brother node $b(h)$ included within $T(E)$.

Observe that we require condition (ii) above to ensure, along with condition (i), that any collapsed node is part of an entire collapsed subtree that is rooted at some node. In other words, (ii) equivalently asserts that $T(E)$ is a *binary tree*.

Now, let us define a set of binary variables as follows:

$$x_h = \begin{cases} 1 & \text{if node } h \text{ is selected as an end node} \\ 0 & \text{otherwise} \end{cases} \quad (16)$$

and consider the following system of constraints:

$$x_h + \sum_{u \in p(h)} x_u = 1, \quad \forall h \in \tilde{T} \quad (17a)$$

$$x_h \text{ binary}, \quad \forall h \in \mathcal{H}. \quad (17b)$$

Proposition 1 below asserts that the system (17) precisely characterizes the set of pruned trees of T in the sense of Definition 1.

Proposition 1 Let $T(E)$ be a pruned tree of T having end nodes E . Define a (binary) vector $x \in \{0,1\}^{|\mathcal{H}|}$ having components $x_h = 1$ if $h \in E$ and $x_h = 0$ otherwise. Then x satisfies system (17). Conversely, consider any feasible solution x to the system (17). Define

$$E \equiv \{h : x_h = 1\} \quad (18a)$$

and let τ be the subtree of T given by

$$\tau \equiv \bigcup_{h \in E} C[h \rightarrow 0]. \quad (18b)$$

Then, τ is a pruned tree of T having end nodes E , i.e., $\tau \equiv T(E)$.

Proof. Let $T(E)$ be a pruned tree of T having end nodes E , and define the binary vector $x \in \{0,1\}^{|\mathcal{H}|}$ to have components $x_h = 1, \forall h \in E$, and $x_h = 0$ otherwise. Let us show that x satisfies (17a). Toward this end, consider any $h \in \tilde{T}$ and let us examine the chain $C[h \rightarrow 0]$ in T . Let v be the first node in this chain that exists within $T(E)$ (see Figure 4.3-1(a)). If $v = h$ itself, then $v \equiv h \in E$, while $E \cap p(h) = \emptyset$ and so, $x_h = 1$ and $x_u = 0, \forall u \in p(h)$. Hence, (17a) holds true. Otherwise, if $v \neq h$, then we know that the nodes on $C[h \rightarrow v)$, in particular, are all collapsed nodes. Let $v' \in s_l(v)$ be the immediate successor of node v that lies on $C[h \rightarrow v]$ (possibly, $v' \equiv h$). Since v' is a collapsed node, we have that $b(v')$ is also a collapsed node by condition (ii) of Definition 1. In fact, together with condition (i) of Definition 1, this implies then that $v \in E$, while $E \cap p(v) = \emptyset$. Consequently, $x_v = 1$ while $x_h = 0$ and $x_u = 0, \forall u \in p(h), u \neq v$. Hence again, we have that (17a) is satisfied.

Conversely, let x be any feasible solution to (17) and define E and τ as in (18a, b). Hence, by construction, τ is a subtree of T that contains the root node 0. Next, let us show that E is precisely the set of end nodes of τ . Again by the construction process (18b), we know that $\{\text{ends of } \tau\} \subseteq E$. On the contrary, if any $e_1 \in E$ is not an end of τ , then there must exist some $e_2 \in E$ for which $e_1 \in C(e_2 \rightarrow 0]$. Moreover, by tracing along successor nodes in T starting from the node e_2 , let $e_3 \in \tilde{T}$ be such that $e_2 \in C[e_3 \rightarrow 0]$ (possibly, $e_3 \equiv e_2$). By applying (17) to e_3 , since $\{e_1, e_2\} \subseteq \{e_3\} \cup p(e_3)$, we must have that $x_{e_1} + x_{e_2} \leq 1$, which is a contradiction because by assumption, $\{e_1, e_2\} \subseteq E$ implies that $x_{e_1} = x_{e_2} = 1$. Hence, $E \equiv \{\text{ends of } \tau\}$, and so, τ satisfies condition (i) of Definition 1.

To complete the proof, let us verify that condition (ii) of Definition 1 holds true by showing that for any node $h \in \tau, h \neq 0$, we also have that $b(h) \in \tau$. Toward this end, consider any $h \in \tau, h \neq 0$, and let v be the immediate predecessor of h on $C(h \rightarrow 0]$. Since $C[v \rightarrow 0] \cap E = \emptyset$, we have from (18a) that

$$x_u = 0, \forall u \in C[v \rightarrow 0]. \quad (19)$$

Now, by tracing successor nodes in T starting from $b(h)$, suppose that we reach an end node $e \in \tilde{T}$ (see Figure 4.3-1(b)), so that

$$C[e \rightarrow 0] \supseteq C[b(h) \rightarrow 0] \equiv b(h) \cup C[v \rightarrow 0]. \quad (20)$$

From (17), we have,

$$x_e + \sum_{u \in C(e \rightarrow b(h))} x_u + x_{b(h)} + \sum_{u \in C[v \rightarrow 0]} x_u = 1. \quad (21)$$

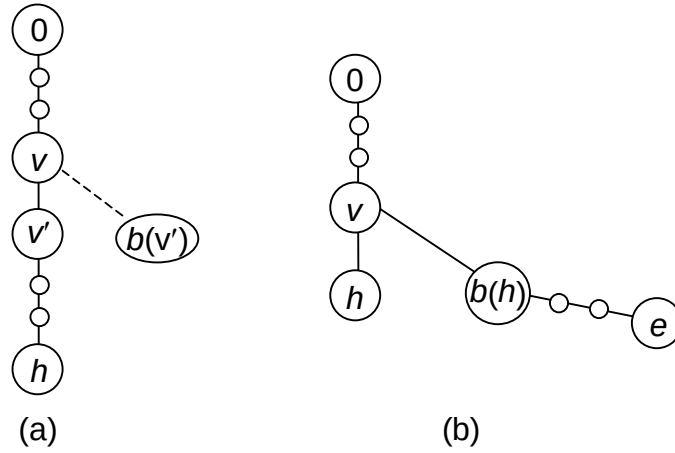


Figure 4.3-1: Illustration for the Proof of Proposition 1.

Therefore, on the contrary, if $b(h) \notin \tau$, then since we have shown above that E is the set of end nodes of τ , we would have that $b(h) \notin E$, and so by (18a), $x_{b(h)} = 0$.

Moreover, we similarly have that $C[e \rightarrow b(h)] \cap \tau = \emptyset$, and so,

$$x_e + \sum_{u \in C(e \rightarrow b(h))} x_u + x_{b(h)} = 0. \quad (22)$$

This together with (19) asserts that the entire left-hand-side in (21) is zero, a contradiction. Hence, $b(h) \in \tau$, and this completes the proof. $\square$

By Proposition 1, the system (17) provides a mathematical structure that affords a great deal of flexibility in designing an “optimal” pruning strategy. Noting that any feasible solution to (17) maps onto a pruned tree $T(E)$ as defined in Proposition 1 via (18), we can ascribe any desired *merit function* $f(h)$ to the end nodes of the prescribed pruned tree, and choose to solve the *pruning problem*

$$\mathbf{PP1:} \text{ Maximize } \left\{ \sum_{h=0}^H f(h)x_h : \text{ system (17)} \right\}. \quad (23)$$

Note that for any $h \in \mathcal{H}$, if h turns out to be an end of $T(E)$ (i.e., $h \in E$) as portended by $x_h = 1$, then by examining the restrictions on the branches in the chain $C[h \rightarrow 0]$ in T , we know precisely the classification characteristics of node h and what survey (or synthetic) population households would reside within node h . Moreover, since any solution to (17) produces a legitimate pruned tree, it would result in a unique classification of household samples into the end nodes $h \in E$ given by (18a). Notice that we have a great deal of flexibility in selecting a merit function, or a combination of merit functions, for formulating (23) as described in Remark 4 below. Moreover, (23) has a special *set partitioning* structure that promotes its solvability (see [Nemhauser and Wolsey (1999)]). Furthermore, from a modeling viewpoint, this approach permits us to include additional required restrictions within the pruning problem (23) as specified in Remark 5 below.

Remark 4 (Alternative Merit Functions). One merit function as used in Section 4.2.3 above can be based on the deviance function as follows, similar to (1), (2):

$$f_1(h) = -\sum_{j=1}^p \sum_{i \in N_h} s_j \left(Y_{ij} - \bar{Y}_{N_h j} \right)^2, \quad \forall h \in \mathcal{H}, \text{ where } \bar{Y}_{N_h j} \equiv \frac{\sum_{i \in N_h} Y_{ij}}{|N_h|}, \quad (24)$$

and where N_h is the subset of the observations $\{1, \dots, n\}$ that would be partitioned into node h were it to become an end node of the pruned tree. (Note that we have used a minus sign in (24) because of the maximization objective in (23). Also, note that f_1

could be evaluated based on the classification of some other independent test sample as in Section 4.2.3.)

Likewise, as in (12), we could augment (24) by imposing a penalty $\alpha \geq 0$ on each end node created. This would yield a merit function

$$f_2(h) = f_1(h) - \alpha \quad (25a)$$

so that

$$\sum_h f_2(h)x_h = \sum_h f_1(h)x_h - \alpha \sum_h x_h$$

where the last term penalizes the total number of end nodes $\sum_h x_h$ by a factor of $\alpha \geq 0$.

Note that in lieu of employing a constant α for each h as in (25a), we can use a factor

$$\alpha_h = \alpha \cdot (\text{depth of node } h) \quad (25b)$$

where the depth of a node h is the number of branches on the chain $C[h \rightarrow 0]$. This would tend to produce relatively more breadth-wise expanded trees, and would yield the following merit function as an alternative to (25a):

$$f_2(h) = f_1(h) - \alpha_h, \forall h \in \mathcal{H}. \quad (25c)$$

Another possible merit function can be based on the total number of trips accrued by some synthetic population L' when it is assigned to the pruned tree. Defining N'_h as the subset of this synthetic population that would be ascribed to node h , given that node h becomes an end node of the prescribed pruned tree, and letting Y'_{ij^*} (with the index j^* defined as for (5)) represent the total number of trips for household i , for each $i \in N'_h$

given that it is matched within node $h \in E$ for a pruned tree $T(E)$, we can construct a merit function

$$f_3(h) = \sum_{i \in N_h} Y'_{ij*}, \forall h \in \mathcal{H}. \quad (26)$$

(Alternatively, we could use average values as in (5) by dividing this by $|N_h|$, depending on our objective.)

We can also handle multiple objectives via a suitable weighting or goal-constrained scheme. For example, we can compose a merit function via (25) and (26) as

$$f_4(h) = \lambda f_2(h) + (1 - \lambda) f_3(h) \text{ for some suitable } 0 < \lambda < 1. \quad (27)$$

Alternatively, letting D^* be the total deviance of the original tree T , we could try composing a pruned tree that maximizes the total assigned trips via the merit function f_3 given by (26), while requiring the total deviance to increase by no more than some permissible tolerance $100\Delta\%$, where $\Delta \geq 0$. This can be achieved by solving the pruning problem

$$\mathbf{PP2:} \text{ Maximize } \left\{ \sum_{h=0}^H f_3(h)x_h : -\sum_{h=0}^H f_1(h)x_h \leq D^* (1 + \Delta), \text{ system (17)} \right\}. \quad (28)$$

Remark 5 (Accommodating Additional Restrictions). Within the context of PP1 or PP2, given respectively by (23) and (28), we could impose various additional restrictions. For example, to restrict the number of end nodes, we could include the constraint

$$\sum_{h=0}^H x_h \leq E_{\max} \text{ for some parameter } E_{\max}, \text{ or more generally,}$$

$$\sum_{h=0}^H \alpha'_h x_h \leq \mu \text{ for some parameter } \mu, \quad (29)$$

for some suitable weights $\alpha'_h > 0, \forall h \in \mathcal{H}$. (For example, we could select $\alpha'_h = \alpha_h$ as given by (25b).) Sometimes, we would like to preserve certain end nodes $E_1 \subseteq \tilde{T}$. For example, there might be end nodes of T that classify either senior citizens or minors, which would naturally generate only few, if any, trips. Such nodes would become prime targets for being collapsed under an objective function that attempts to maximize the total number of assigned trips subject to additional constraints as in (28) or (29). However, because of the special nature of these end nodes, from a statistical record point of view, we might prefer to preserve such nodes in the pruned tree. In such a case, we can simply impose the restriction $x_h = 1, \quad \forall \quad h \in E_1$. Likewise, to conserve modeling and solution effort, we might wish to enforce that certain nodes E_0 of T do *not* turn out to be end nodes (i.e., they are either collapsed, or are preserved as non-terminal nodes of the prescribed pruned tree). This can be accommodated by setting $x_h = 0, \forall h \in E_0$. Several other types of desired restrictions and objectives can be similarly modeled within such optimal pruning problem formulations. In our study, we shall investigate and empirically evaluate several such delineated problems.

4.4 Overall Activity Generator Module

The overall process for the Activity Generator Module of TRANSIMS can be described as follows (see [Los Alamos National Laboratory (2002a)] for more details).

1. Take the given survey population (referred to as L above) and record the sequences of types of activities for each observation, stripped of any specific locations. Also, identify the demographic attributes X_k , $k = 1, \dots, K$, and their ranges over the given population, and compute the measures Y_{ij} for each $i \in L$, $j = 1, \dots, p$, as well as estimate the weights s_j for each $j = 1, \dots, p$.
2. Use the tree-growing and pruning/reduction procedures described in Sections 4.1-4.3 to arrive at a binary tree T to be used along with its terminal nodes $\tilde{T}$. Identify the classification scheme associated with each terminal node in $\tilde{T}$, as given by the restrictions on the branches leading from this node to the root of T (follow the predecessor DOWN($\cdot$) labels for this purpose).

Then, given any synthesized household, assign it to the appropriate terminal node $N \in \tilde{T}$ based on its demographic characteristics. Randomly match this synthesized household with a survey household in N using the discrete probability density function $p(i) = w_i / \sum_{j \in N} w_j$, $\forall i \in N$, where w_i is some (importance) weight ascribed to the observation i in N . Note that the matched survey household might not be perfectly compatible with the synthesized household. Hence, we may modify the former (or try a different match) as described in [Los Alamos National Laboratory (2002a)].

3. Because the matching at Step 2 is based on demographics, it is assumed that the list of activities for any person in the matched survey household is appropriate for the

corresponding person in the synthesized household. However, the locations for these activities need to be generated. Using the principal origin and destination locations of the synthesized household as the base input, generate locations for the activities as in [Los Alamos National Laboratory (2002a)] using a *discrete choice model* to first select appropriate *zones* for all the activities, and then using *land-use* information to specify the actual *locations within zones* for each of the activities.

4. Prescribe as the final output a list of the set of participants from the synthesized household for each of the activity trips, providing information on ranges for the start-time, end-time, and the duration for each activity (see [Los Alamos National Laboratory (2002a)] for details).

4.5 Implementation and Results

In Section 4.3, we have described a new concept for formulating optimization models to conduct the pruning of classification trees obtained via the CART algorithm. Various merit or objective functions, along with desired side-constraints, can be used within this context. Note that this idea is generally applicable to any situation where the CART procedure is employed.

In particular, within the context of the Activity Generator Module of TRANSIMS, we investigated the use of different objective functions and side-constraints as delineated in Section 4.3 to both develop as well as optimally prune classification trees. We implemented the modified CART algorithm using JAVA that utilizes the MySQL server

version 4.0 (MySQL) and lp_solve 2.0 (M. Berkelaar (1997)), which is written in JAVA under the Lesser GNU Public License. This program, CART-VT, allows the user to load a survey data in an Excel file format. We loaded 3,470 household survey data, which were available from studies conducted on the Portland transportation network (see [Los Alamos National Laboratory (2002b)]), into CART-VT as shown in Figure 4.5-1. For illustrative purpose, we used 23,656 synthetic households that pertain to the Blacksburg transportation network.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	
	Househ...	T_Home	T_Work	T_Shop	T_Sch...	T_Visit	T_Other	N_Non...	HHSIZE	VEHIC...	INCOME	ALT5	A5TO17	A26T...	H
	Y1	Y2	Y3	Y4	Y5	Y6	Y7		X1	X2	X3	X4	X5	X6	
1															
2															
3	200002	918	417	0	0	0	180	3	1	1	14	0	0	1	
4	200009	1395	0	188	0	0	0	2	1	0	1	0	0	0	
5	200010	2275	385	210	0	0	161	11	2	2	11	0	0	1	
6	200015	1620	0	0	0	0	0	0	1	0	14	0	0	0	
7	200020	1310	0	0	0	0	225	2	1	0	2	0	0	0	
8	200029	2970	770	175	0	10	700	14	3	3	4	0	0	1	
9	200030	1560	0	0	0	0	1	1	1	0	3	0	0	0	
10	200037	4260	475	20	365	0	945	16	4	2	13	1	1	2	
11	200045	3060	1180	0	375	0	166	6	3	2	13	0	1	1	
12	200055	1765	1145	30	0	35	126	10	2	2	13	0	0	0	
13	200056	6675	285	0	0	0	946	6	5	1	10	0	4	1	
14	200057	7470	1260	0	2070	0	265	14	7	2	11	0	5	2	
15	200058	4634	540	187	0	4	724	14	4	2	13	1	1	1	
16	200060	2560	0	75	0	0	495	6	2	2	10	0	0	1	
17	200061	3085	0	30	0	0	80	4	2	3	14	0	0	0	
18	200067	4115	540	210	0	15	1265	18	4	2	11	2	0	2	
19	200069	4670	420	180	240	170	300	9	3	2	4	1	0	0	
20	200070	1360	0	0	0	88	70	5	1	1	4	0	0	0	
21	200077	2535	355	71	0	0	160	6	2	2	13	0	0	1	
22	200087	2385	0	0	0	0	765	4	2	2	6	0	0	2	
23	200088	3140	723	0	0	0	846	8	3	2	14	0	1	2	
24	200089	2610	1065	0	605	0	255	12	3	3	13	1	0	2	
25	200090	2650	0	325	0	0	200	5	2	1	5	0	0	0	
26	200093	990	48	0	0	0	0	1	1	1	6	0	0	0	
27	200098	915	0	125	480	0	60	4	1	1	2	0	0	1	
28	200100	1920	1035	35	60	0	10	7	2	2	5	0	0	1	
29	200107	1410	0	0	0	0	180	1	1	1	4	0	0	0	
30	200120	3869	810	0	0	545	945	12	4	2	13	0	2	2	
31	200124	3120	345	70	800	0	240	12	3	1	8	0	1	2	
32	200126	1395	0	0	0	0	210	1	1	1	2	0	0	0	

Figure 4.5-1: A screenshot of CART-VT with the survey data in Excel format.

The user can select dependent and independent variables for constructing the CART from the headers of the corresponding columns in the survey data. Figure 4.5-2 depicts the various variables available for selection as pertaining to our survey data.

The image shows a dialog box titled "CART Tree Preferences" with three tabs: "Variables", "Tree Options", and "DB Options". The "Variables" tab is active, displaying a list of variables and checkboxes for their selection as dependent or independent variables.

Variables	Dependent	Independent
Household	☐	☐
T_Home	☒	☐
T_Work	☒	☐
T_Shop	☒	☐
T_School	☒	☐
T_Visit	☒	☐
T_Other	☒	☐
N_Non_home	☒	☐
HHSIZE	☐	☒
VEHICLES	☐	☒
INCOME	☐	☒
ALT5	☐	☒
A5T017	☐	☒
A26T045	☐	☒
HHAGE	☐	☒
WORKERS	☐	☒
HDENSTY	☐	☒

At the bottom of the dialog box are two buttons: "Plot" and "Cancel".

Figure 4.5-2: Selection of variables for constructing the CART.

Once the desired variables are specified, the user has an option to select the CART parameters as discussed in Section 4.2, such as $N_{\min}$ (which is referred to as “Min. Node Size” in CART-VT) and the multiplier of the total deviance $D(L)$ to produce the threshold value β (which is referred to as “Min. Node Deviance” in CART-VT). In addition, the user can select to use one of two available pruning schemes: the Cross-Validation Scheme and the Independent Test Sample Scheme, as shown in Figure 4.5-3. We constructed the CART using $N_{\min}=10$ and the multiplier of the total deviance equal to 0.01. This was accomplished in about 34 minutes for our survey data on a Dell 400SC 2.8 GHz computer having 1 GB of memory, using Redhat Linux 7.3, and it produced a CART having 100 chains and 199 nodes as shown in Figure 4.5-4.

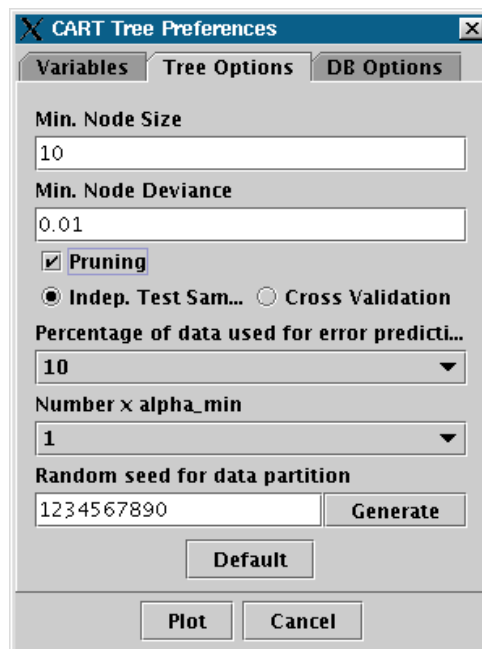


Figure 4.5-3: Pruning method selection for constructing the CART.

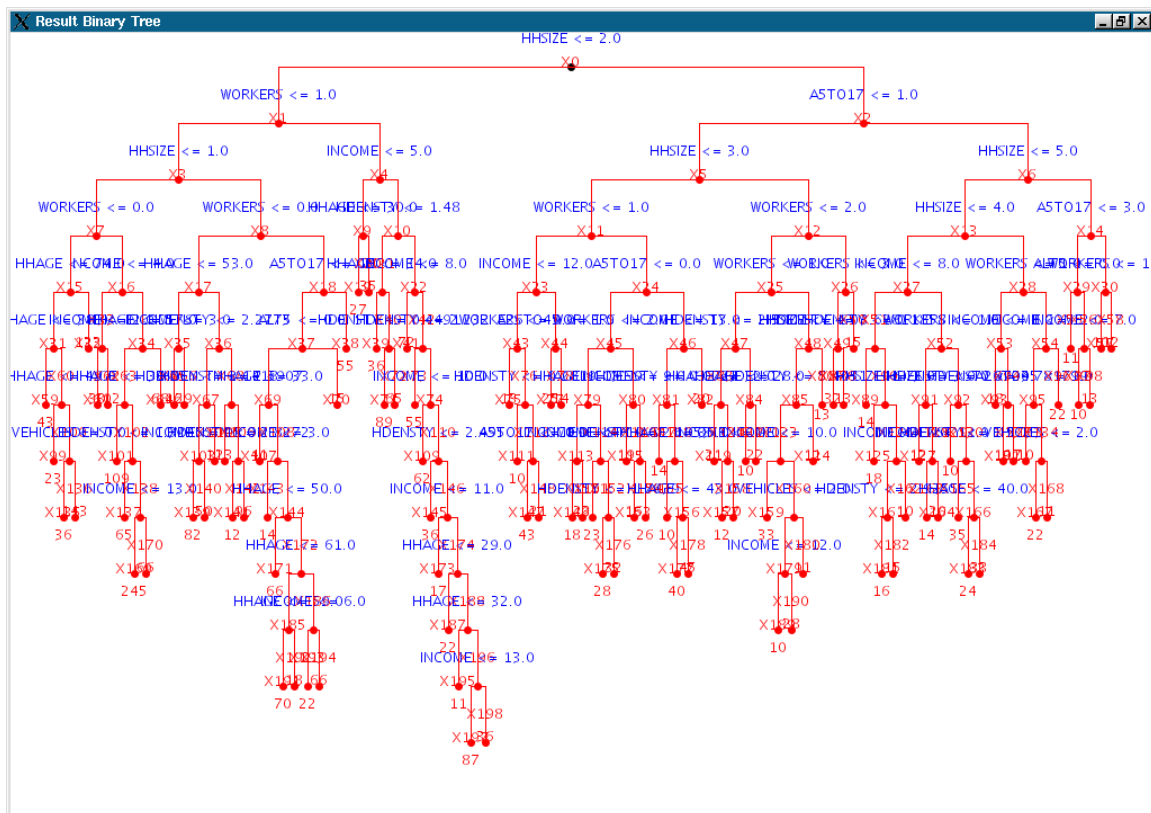


Figure 4.5-4: The CART produced by using $N_{\min}=10$ and the multiplier of the total deviance equal to 0.01.

We next implemented CART-VT along with various options of objective functions and desired side-constraints as shown in Figure 4.5-5 and Figure 4.5-6, respectively. The user can select to minimize the total number of end nodes, minimize the maximum depth of the end nodes, minimize the sum over all the end nodes of the ratios of the total number of synthetic households and the survey households that fall into the same end node, minimize the maximum over the end nodes of the ratio of the total number of synthetic households and the survey households that fall into the same end node, minimize the total deviance, maximize the total sum of some selected variable over the end nodes (such as the total number of trips), or optimize any weighted sum of the available objective functions where the total weight must sum to 1.

In TRANSIMS, we use the CART algorithm to select a suitable survey household to match any given synthetic household in order to produce the activity pattern for the members of the synthetic household. We might not want to produce many duplications of the same activity pattern. Therefore, this motivates the foregoing objective functions of minimizing the sum over all the end nodes of the ratios of the total number of synthetic households and the survey households that fall into the same end node, or minimizing the maximum of these ratios.

The side-constraints that are proposed for CART-VT impose a bound on the total number of end nodes, a bound on the depth of each end node, a bound on the ratio of the deviance of the new tree and that of the original tree, and a bound on the ratio between the total number of synthetic households and the survey households that fall into the same end node (for each end node). The user can select a lower bound, an upper bound, or both for each side-constraint. In addition, the user can select to preserve some special

end nodes or enforce certain end nodes that are produced in the original tree to not be end nodes in the optimally pruned classification tree. This selection is available as “Constraint on each end node” in the Constraints tab as shown in Figure 4.5-6. After selecting the desired objective functions and side-constraints, the user can generate and solve the model to obtain an optimally pruned classification tree.

The dialog box is titled "Selective Pruning Strategy Preferences" and has three tabs: "Objective Functions", "Constraints", and "Population Synthesizer File". The "Objective Functions" tab is selected. It contains a list of objective functions with checkboxes and a corresponding weight column. The instructions state: "Please select objective function(s) and specify the corresponding weight(s) that must sum up to 1." The objective functions and their weights are:

Objective Function	Weight (between 0 and 1)
☐ Minimize the total number of end nodes.	0.5
☐ Minimize the number of branches from the root node to the end nodes.	0.5
☐ Minimize the sum over all the end nodes of the ratios of the total number of synthetic households and the survey households that fall into the same end node.	0.5
☐ Minimize the maximum over the end nodes of the ratio between the total number of synthetic households and the survey households that fall into the same end node.	0.5
☐ Minimize the total deviance of the new tree.	0.5
☐ Maximize the total sum of the selected variable.	0.5

At the bottom right, there is a "Variables" button with a value of 0.5. At the bottom center, there are "Generate IP" and "Cancel" buttons.

Figure 4.5-5: Various objective functions for optimally pruning the classification tree.

The dialog box is titled "Selective Pruning Strategy Preferences" and has three tabs: "Objective Functions", "Constraints", and "Population Synthesizer File". The "Constraints" tab is selected. It contains a list of side-constraints with checkboxes and a corresponding parameter column. The instructions state: "Warning, certain constraints may lead to infeasibility." The side-constraints and their parameters are:

Constraint	Parameter
☐ Lower upper bound on the total number of end nodes	0
☐ Lower upper bound on the number of branches from the root node to the end nodes	5
☐ Lower upper bound on the ratio of the deviance of the end nodes of the new tree and that of the original tree	0.1
☐ Constraint on each end node	Parameters
☐ The ratio between the total number of synthetic households and the survey households that fall into the same end node	100

At the bottom center, there are "Generate IP" and "Cancel" buttons.

Figure 4.5-6: Various side-constraints for optimally pruning the classification tree.

We investigated different models by composing suitable objective functions with various appropriate combinations of side-constraints. For example, it is inappropriate to simply minimize the total number of end nodes, because the resulting optimally pruned classification tree will then contain only the root node. Therefore, we should incorporate additional side-constraints that would preserve some end nodes to a desirable extent in the tree. By adding the side-constraint that the total number of end nodes should be

greater than or equal to six, for example, the resulting optimally pruned classification tree contains exactly six end nodes as shown in Figure 4.5-7 where the objective function of -6 corresponds to maximizing the negative of the specified objective function. (Note that all the results given below utilize such a transformed equivalent maximization objective function.)

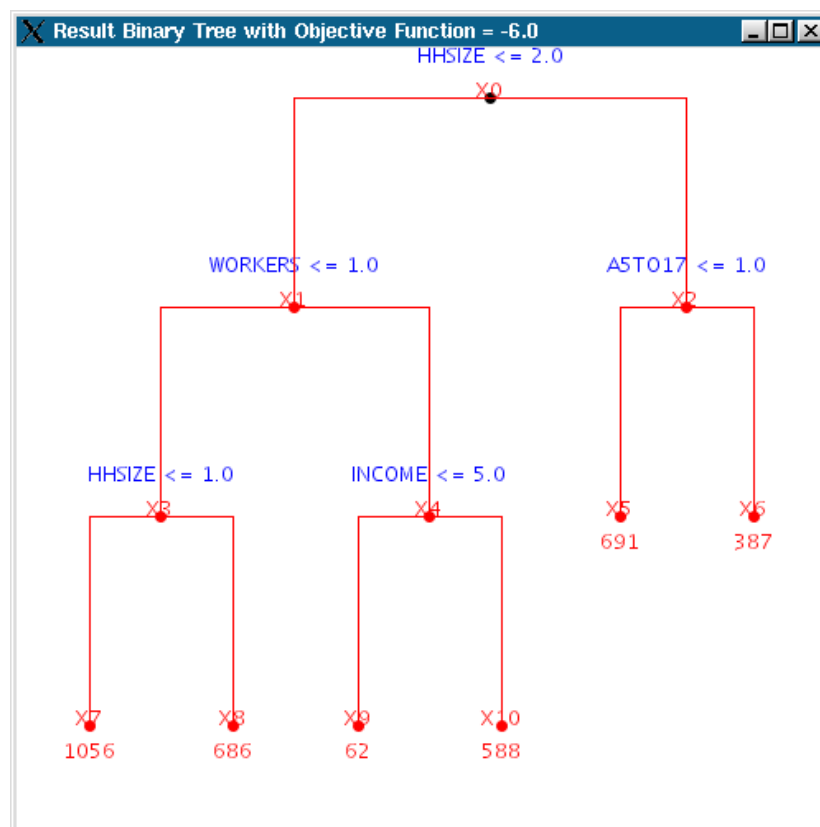


Figure 4.5-7: The optimally pruned classification tree for minimizing the number of end nodes subject to this number being at least six.

Alternatively, suppose that we are interested in preserving a certain original end node that contains some special demographic characteristic, for example, householders comprised of only two workers having a total income of less than \$50,000 and each being of age less than 30 years. This special demographic characteristic falls into Node 19. The optimally pruned classification tree is shown in Figure 4.5-8 where the (maximization)

objective function value is -5. Note here that we are minimizing the total number of end nodes while preserving Node 19. Therefore, the brother of Node 19, which is Node 20 ($b(19) \equiv 20$), must be an end node according to Condition (ii) in Definition 1. Since Node 19 is in the pruned tree, Node 9 and its brother Node 10 must also be in the pruned tree according to conditions (i) and (ii). Likewise Node 4 and its brother Node 3 must be in the pruned tree, as well as Node 1 and its brother Node 2. This results in the optimally pruned classification tree as depicted in Figure 4.5-8. This same pruned tree is obtained when the objective is to minimize the depth of the end nodes while preserving Node 19.

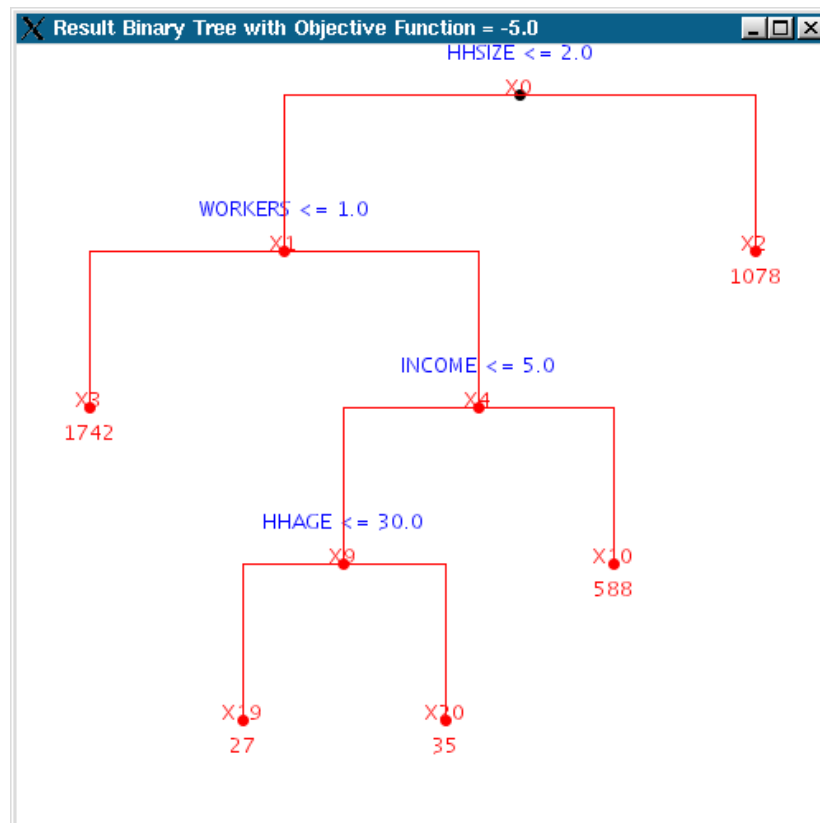


Figure 4.5-8: The optimally pruned classification tree for minimizing the total number of end nodes while preserving Node 19 and for minimizing the depth of the end nodes while preserving Node 19.

As discussed previously, we use the CART algorithm to select a suitable survey household to match any given synthetic household in order to produce the activity pattern

for the members of the synthetic household. Therefore, we would minimize the total ratio between the total number of synthetic households and the survey households that fall into the same end node. The synthetic household data that we used for experimentation is from the Blacksburg transportation network and it contains 23,656 households. If there are no additional side-constraints imposed, the resulting tree would contain only the root node, which is not acceptable. Thus, we put a bound on the total number of end nodes to be greater than or equal to five. The resulting tree, having a (maximization) objective value of -19, is shown in Figure 4.5-9.

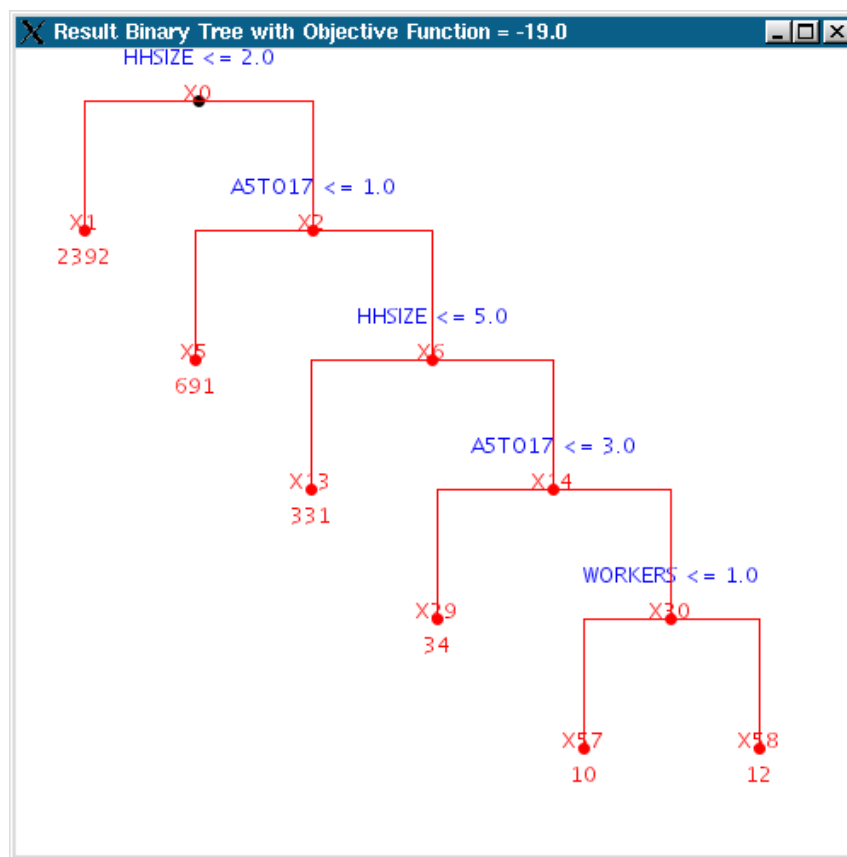


Figure 4.5-9: The optimally pruned classification tree for minimizing the sum over all the end nodes of the ratios of the total number of synthetic households and the survey households that fall into the same end node subject to the total number of end nodes being greater than or equal to five.

Another objective function that might be of interest from a statistical viewpoint is to minimize the total deviance of the new tree. The optimally pruned classification tree

obtained by using this objective function without any constraints is exactly the same as the original tree, because the original tree is the minimal deviance tree. Figure 4.5-10 depicts this tree. However, if we add the side-constraint that the ratio between the total number of synthetic households and the survey households that fall into the same end node be less than or equal to seven for all end nodes, the resulting tree obtained is depicted in Figure 4.5-11. This tree provides a matching that does not duplicate the activity patterns more than seven times. In addition, we can add an additional side-constraint to limit the depth of the end node to be less than or equal to five, which produces the optimally pruned classification tree as shown in Figure 4.5-12.

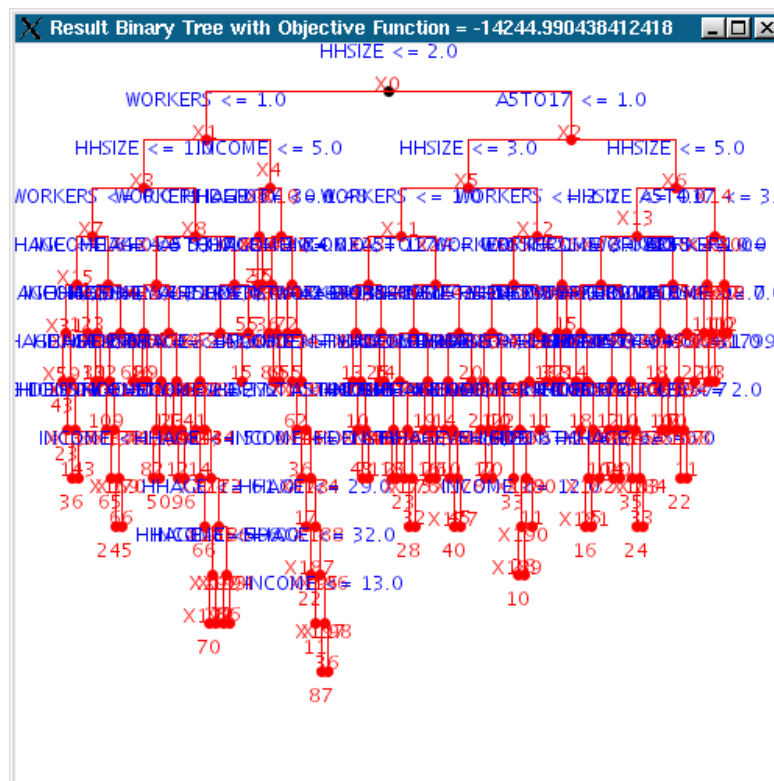


Figure 4.5-10: The optimally pruned classification tree for minimizing the total deviance of the new tree.

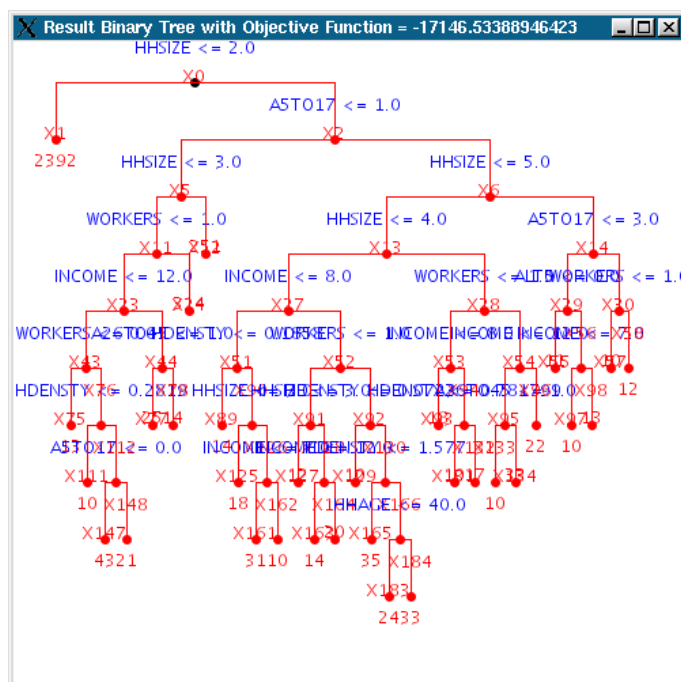


Figure 4.5-11: The optimally pruned classification tree for minimizing the total deviance of the new tree subject to the ratio between the total number of synthetic households and the survey households that fall into the same end node being less than or equal to seven for all end nodes.

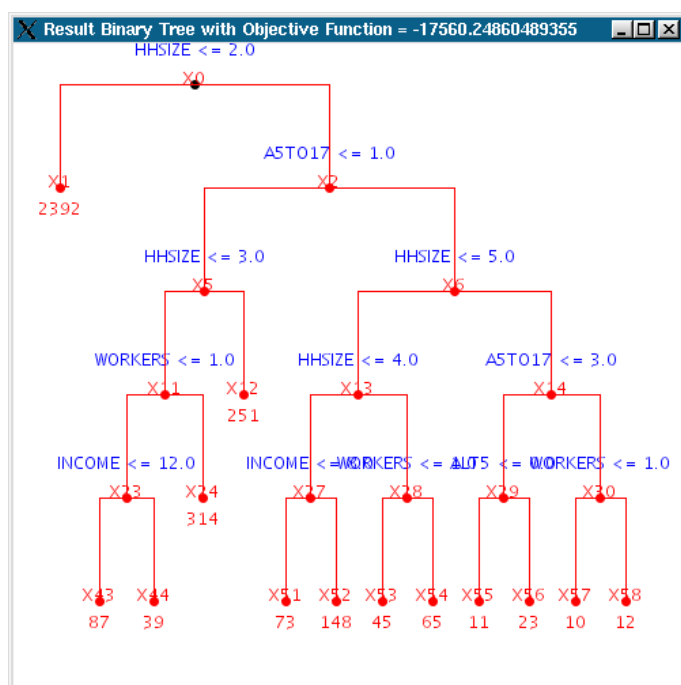


Figure 4.5-12: The optimally pruned classification tree for minimizing the total deviance of the new tree subject to the ratio between the total number of synthetic households and the survey households that fall into the same end node being less than or equal to seven for all end nodes and the depth of each end node being less than or equal to five.

The user can combine different objective functions with desired associated weights to reflect their relation importance. We combined the two objective functions of minimizing the total deviance of the new tree, and maximizing the total number of trips corresponding to the end nodes, where the weight for each objective function was taken as 0.5. In addition, we utilized two side-constraints to bound the ratio between the total number of synthetic households and the survey households that fall into the same end node to be less than or equal to seven for each end node, and to limit the depth or level of the end nodes to be less than or equal to four. The resulting tree is displayed in Figure 4.5-13. It is interesting to compare this tree with those depicted in Figure 4.5-11 and Figure 4.5-12 for related objective functions and side-constraints.

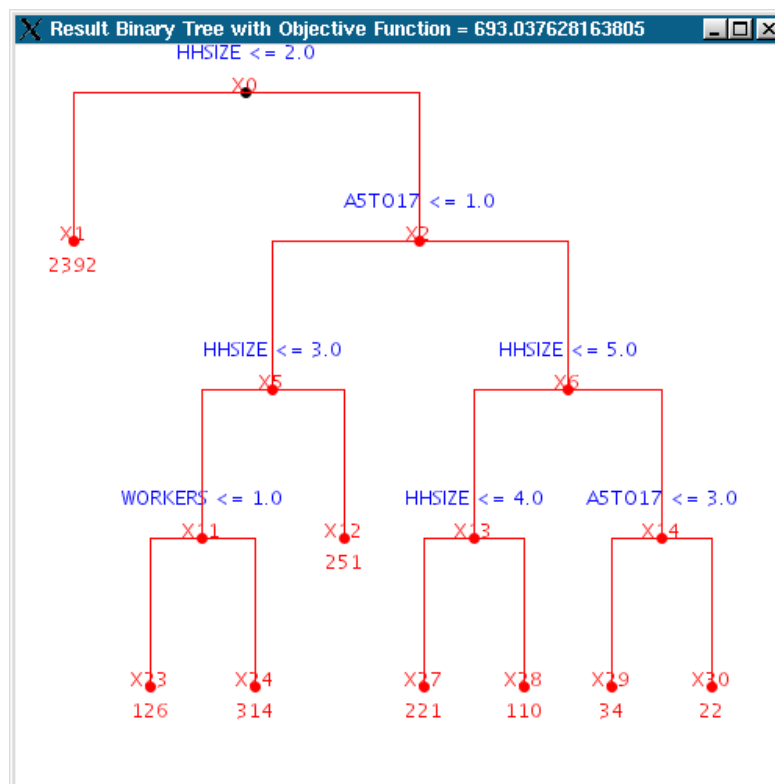


Figure 4.5-13: The optimally pruned classification tree for combined objective functions of minimizing the total deviance of the new tree and maximizing the total number of trips corresponding to the end nodes, where the weight for each objective function was taken as 0.5, subject to the ratio between the total number of synthetic households and the survey households that fall into the same end node being less than or equal to seven for each end node, and with the depth or level of each end nodes being less than or equal to four.

4.6 Summary and Conclusions and Recommendations for Future Research

The Classification and Regression Tree (CART) algorithm (see [Breiman *et al.* (1984)]) is extensively used to conduct analyses of survey data to provide information regarding the correlation between some independent household demographic X -variables and some dependent activity time or trip time Y -variables for these households. This classification and correlation process permits the user to explore different combinations of dependent and independent variables. Using a novel set partitioning formulation, we have proposed a novel optimal pruning strategy that allows users to have control or direction on how to achieve some desired objectives in this trip or activity generation process, or while incorporating certain side-constraints pertaining to the desired structure of the resulting classification tree. We have provided the mathematical structure for characterizing and computing such optimally pruned binary trees. The developed mathematical model permits us to explore a variety of combinations of appropriate objective functions and side-constraints in constructing the pruned tree. Our JAVA program, CART-VT, utilizing the MySQL database server, provides an option of using six different objective functions such as minimizing the total number of end nodes, minimizing the maximum depth of the end nodes, minimizing the sum over all the end nodes of the ratios of the total number of synthetic households and the survey households that fall into the same end node, minimizing the maximum over the end nodes of the ratio between the total number of synthetic households and the survey households that fall into the same end node, minimizing the total deviance, maximizing the total sum of some selected variables over the end nodes (such as the total number of trips), or optimizing

any weighted combination of these objective functions, where the total weight must sum to 1. The side-constraints that are available in CART-VT are a bound on the total number of end nodes, a bound on the depth of each end node, a bound on the ratio of the deviance of the new tree and that of the original tree, and a bound on the ratio between the total number of synthetic households and the survey households that fall into the same end node (for each end node). The user can select a lower bound, an upper bound, or both for each side-constraint. In addition, the user can select to preserve some special original end nodes or enforce certain original end nodes not to be end nodes in the optimally pruned classification tree. Note that this idea is generally applicable to any situation where the CART procedure is employed. This method provides a great deal of flexibility in designing the pruned tree by optimizing any desired combination of objective functions subject to appropriate the side-constraints.

For computational experimentations, we used the 3,470 household survey data that is available from studies conducted on the Portland transportation network (see [Los Alamos National Laboratory (2002b)]). For illustrative purpose, the synthetic household data that we used pertains to the Blacksburg transportation network and it contains 23,656 households. We investigated the use of different objective functions as well as side-constraints that are available in CART-VT to produce the optimally pruned classification trees. The resulting CART was obtained for this data set in under 34 CPU minutes, and the optimal pruned classification tree was obtained in 5 minutes on a Dell 400SC 2.8 GHz computer having 1 GB of memory and using Redhat Linux 7.3. Moreover, the varying structures of these trees for different combinations of objective functions and constraints provide interesting insights and support the efficacy of the

proposed approach. For future studies, we suggest testing this optimal pruning strategy in different contexts where the CART algorithm is applied in practice, where this might require additional merit functions and/or side-constraints that are suitable to the particular applications.

References

1. R. K. Ahuja, T. L. Magnanti and J. B. Orlin, *Network flows: Theory, Algorithms, and Applications*, Prentice-Hall, New Jersey (1993).
2. C. Barrett, R. Jacob and M. Marathe, "Formal-Language-Constrained Path Problems," *SIAM Journal on Computing*, v 30 n 3. p 809-837 (2001).
3. M. S. Bazaraa, J. J. Jarvis and H. D. Sherali, *Linear Programming and Network Flows*, second edition, John Wiley and Sons, New York, NY (1990).
4. M. Berkelaar, lp_solve 2.0,
<http://www.cs.wustl.edu/~javagrp/help/LinearProgramming.html> (1997).
5. L. Breiman, J. H. Friedman, L. A. Olshen and C. J. Stone, *Classification and Regression Trees*, Wadsworth International, Belmont, CA (1984).
6. C. Cappelli, F. Mola and R. Siciliano, "An Alternating Pruning Method Based on the Impurity Complexity Measure," COMPSTAT. Proceedings in Computational Statistics, p 221-226 (1998).
7. D. I. A. Cohen, *Introduction to Computer Theory*, second edition, John Wiley and Sons, New York, NY (1990).
8. K. L. Cook and E. Halsey, "The Shortest Route through a Network with Time-Dependent Intermodal Transit Times," *Journal of Mathematical Analysis and Application*, v 14. p 493-498 (1966).
9. E. W. Dijkstra, "A Note on Two Problems in Connection with Graphs," *Numerical Mathematics*, v 1. p 269-271 (1959).

10. S. E. Dreyfus, "An Appraisal of Some Shortest Path Algorithms," *Operations Research*, v 17. p 395-412 (1969).
11. EMME/2 User's Manual, Release 9, Turn Penalty function, p 3-11 (1998).
12. F. Glover, D. Klingman and N. Phillips, "A New Polynomially Bounded Shortest Path Algorithm," *Operations Research*, v 33 n 1. p 65-73 (1985a).
13. F. Glover, D. Klingman, N. Phillips and R. Schneider, "New Polynomial Shortest Path Algorithm and their Computational Attributes," *Management Science*, v 31 n 9. p 1106-1128 (1985b).
14. H. J. Halpern, "Shortest Route with Time Dependent Length of Edges and Limited Delay Possibilities in Nodes," *Zeitschrift fur Operations Research*, v 21. p 117-124 (1977).
15. P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions of Systems Science and Cybernetics SSC*, v 4 n 2. p 100-107 (1968).
16. R. Hassin, "Approximation Schemes for the Restricted Shortest Path Problems," *Mathematics of Operations Research*, v 17 n 1. p 36-42 (1992).
17. R. Jacob, M. Marathe and K. Nagel, "A Computational Study of Routing Algorithms for Realistic Transportation Networks," *Los Alamos National Laboratory Report Series LA-UR-98-2249* (1998).
18. D. E. Kaufman and R. L. Smith, "Fastest Paths in Time-Dependent Networks for IVHS Applications," *IVHS Journal*, v 1. p 1-11 (1993).

19. Los Alamos National Laboratory, Transportation Analysis Simulation System (TRANSIMS) Version: TRANSIMS-LANL-3.0, NM (2002a).
20. Los Alamos National Laboratory, Transportation Analysis Simulation System (TRANSIMS) Portland Study Reports, NM (2002b).
21. A. O. Mendelzon and P. T. Wood, "Finding Regular Simple Paths in Graph Databases," *SIAM Journal on Computing*, v 24 n 6. p 1235-1258 (1995).
22. F. Mola and R. Siciliano, "A Fast Splitting Procedure for Classification Trees," *Statistics and Computing*, v 7. p 208-216 (1997).
23. MySQL, <http://www.mysql.com/>.
24. G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*, second edition, John Wiley & Sons, New York, N.Y. (1999).
25. A. Orda and R. Rom, "Shortest-Path and Minimum-Delay Algorithms in Networks with Time-Dependent Edge-Lengths," *Journal of the Association for Computing Machinery*, v 37. p 607-625 (1990).
26. G. Ramalingam and T. Reps, "An Incremental Algorithm for a Generalization of the Shortest-Path Problem," *Journal of Algorithms*, v 21. p 267-305 (1996).
27. J.-F. Romeuf, "Shortest Path under Rational Constraints," *Information Processing Letters*, v 28. p 245-248 (1988).
28. R. Sedgewick and J. S. Vitter, "Shortest Paths in Euclidean Graphs," *Algorithmica*, v 1. p 31-48 (1986).

29. H. D. Sherali, "On the Equivalence Between Some Shortest Path Algorithms," *Operations Research Letters*, v 10. p 61-65 (1991).
30. H. D. Sherali, A. G. Hobeika and S. Kangwalklai, "Time-Dependent Label-Constrained Shortest Path Problems with Applications to TRANSIMS: A Transportation Planning Model," *Transportation Science*, v 37 n 3. p 278-293 (2003).
31. P. C. Taylor and B. W. Siverman, "Block Diagrams and Splitting Criteria for Classification Trees," *Statistics and Computing*, v 3. p 147-161 (1993).
32. U.S. Department of Transportation, TMIP: Data Collection in the Portland, Oregon, Metropolitan Area Case Study, Washington, D.C. (1996).