

ABSTRACT

Title of dissertation: ON FINDING PATHS AND FLOWS IN
MULTICRITERIA, STOCHASTIC AND
TIME-VARYING NETWORKS

Sathaporn Opasanon, Doctor of Philosophy, 2004

Dissertation directed by: Professor Elise Miller -Hooks
Department of Civil and Environmental Engineering

This dissertation addresses two classes of network flow problems in networks with multiple, stochastic and time-varying attributes. The first problem class is concerned with providing routing instructions with the ability to make updated decisions as information about travel conditions is revealed for individual travelers in a transportation network. Three exact algorithms are presented for identifying all or a subset of the adaptive Pareto-optimal solutions with respect to the expected value of each criterion from each node to a desired destination for each departure time in the period of interest.

The second problem class is concerned with problems of determining the optimal set of *a priori* path flows for evacuation in capacitated networks are addressed, where the time-dependent and stochastic nature of arc attributes and capacities inherent in these problems is explicitly considered. The concept of Safest Escape is formulated for developing egress instructions. An exact algorithm is proposed to determine the pattern of

flow that maximizes the minimum path probability of successful arrival of supply at the sink

While the Safest Escape problem considers stochastic, time-varying capacities, arc travel times, while time-varying, are deterministic quantities. Explicit consideration of stochastic and time-varying travel times makes the SEscape problem and other related problems significantly more difficult. A meta-heuristic based on the principles of genetic algorithms is developed for determining optimal path flows with respect to several problems in dynamic networks, where arc traversal times and capacities are random variables with probability mass functions that vary with time. The proposed genetic algorithm is extended for use in more difficult, stochastic, time-varying and multicriteria, capacitated networks, for which no exact, efficient algorithms exist. Several objectives may be simultaneously considered in determining the optimal flow pattern: minimize total time, maximize expected flow and maximize the minimum path probability of successful arrival at the sink (the objective of the SEscape problem). Numerical experiments are conducted to assess the performance of all proposed approaches.

**ON FINDING PATHS AND FLOWS IN MULTICRITERIA,
STOCHASTIC AND TIME-VARYING NETWORKS**

by

Sathaporn Opananon

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2004

Advisory Committee:

Professor Elise Miller-Hooks, Chair
Professor Hani S. Mahmassani
Professor Bruce L. Golden
Professor Ali Haghani
Professor Gregory B. Baecher

©Copyright by
Sathaporn Opasanon
2004

ACKNOWLEDGMENTS

I wish to express my great gratitude to Dr. Elise Miller-Hooks who serves as my advisor and mentor throughout my studies at Pennsylvania State University and University of Maryland. I am sincerely grateful to Dr. Miller-Hooks for the opportunity she has given me, for inspiration and encouragement, for her patience, her accessibility and her invaluable guidance, which play a critical role in the completion of this dissertation. I have learned a great deal from her in both academic and personal life. Without her, I would not have come this far.

I would like to thank Dr. Konstadinos Goulias, Dr. Ageliki Elefteriadou, Dr. Paul P. Jovanis and Dr. A. Ravi Ravindran who served on my thesis committee at Pennsylvania State University. I am greatly honored to have Professors Hani S. Mahmassani, Bruce L. Golden, Ali Haghani and Gregory B. Baecher in my doctoral committee at University of Maryland. Their inestimable advice during the defense contributes to the perfection of my dissertation. I would especially like to thank Professor Mahmassani for his comments on the entire dissertation.

I would like to acknowledge all the friends and colleagues at Pennsylvania State University and University of Maryland. Specifically thank to Nateekool Kriangchaiporn for making State College my home and Hathairat Maneetes for warm support. My thanks also go to Dr. Baiyu Yang and Dr. Hao Tang for their help and friendship. I would like to thank Harsh Dhundia, Minseok and all the office mates who make my life in College Park enjoyable.

Finally, I would like to express my deepest appreciation to my parents for their endless love and support, which gave me idea, motivation, strength and power to conquer any obstacle.

TABLE OF CONTENTS

List of Tables.....	viii
List of Figures.....	ix
Chapter 1. Introduction.....	1
1.1. Motivation.....	1
1.2. Problem Class I.....	3
1.3. Problem Class II.....	6
1.4. Contributions.....	11
Chapter 2. Adaptive Pareto-Optimal Path Strategies.....	13
2.1. Introduction.....	13
2.2. Literature Review.....	15
2.2.1. Single Criterion Shortest Path Problems.....	15
2.2.2. Multicriteria Optimal Path Problems.....	17
2.3. Pareto-Optimal Hyperpaths in STV Networks.....	21
2.4. Network Notation and Problem Definitions.....	27
2.5. Solution Approaches.....	31
2.5.1. The Adaptive Pareto-Optimal Strategy (APS) Algorithm.....	32
2.5.2. The Adaptive Least Expected Disutility Strategy I (ALEDS I) Algorithm	43
2.5.3. The Adaptive Least Expected Disutility Strategy II (ALEDS II) Algorithm.....	47

2.6. Notes on Algorithm Implementation.....	52
2.7. Computational Experiments.....	53
2.7.1. Experimental Design.....	53
2.7.2. Average Run Times of the APS Algorithm.....	54
2.7.3. Average Run Times of the ALEDS I and II Algorithms.....	55
2.8. Conclusions.....	58
Chapter 3. Adjustable Preference Path Strategies.....	60
3.1. Introduction.....	60
3.2. Network Notation and Problem Definition.....	63
3.3. The Adjustable Preference Path Strategy (APPS) Algorithm.....	64
3.4. Illustrative Example Problem.....	68
3.5. Numerical Experiments.....	73
3.5.1. Experimental Design.....	74
3.5.2. Results and Discussion.....	74
3.6. Conclusions.....	76
Chapter 4. The Safest Escape Problem.....	77
4.1. Introduction.....	77
4.2. Literature Review.....	80
4.3. Conceptual Framework.....	87
4.3.1. Expectation and Path Flows.....	88
4.3.2. Probabilities of Successful Path Traversal.....	91

4.4. Safest Escape.....	92
4.5. Network Notation and Problem Formulation.....	95
4.6. Solution Approach.....	98
4.6.1. The PTD residual network.....	98
4.6.2. SEscape algorithm.....	100
4.6.3. Maximum Probability Path (MPP) algorithm.....	104
4.6.4. Proof of the SEscape algorithm.....	107
4.7. Illustrative Example.....	112
4.8. Numerical Experiments.....	117
4.8.1. Experimental Design.....	117
4.8.2. Experimental Results.....	118
4.9. Conclusions.....	121
Chapter 5. Heuristics for MSTV Capacitated Networks.....	123
5.1. Introduction.....	123
5.2. A Genetic Algorithm for Deterministic, TimeV arying Networks.....	130
5.2.1. Network Notation and Problem Definition.....	131
5.2.2. Genetic Algorithm.....	131
5.2.3. Illustrative Example.....	140
5.2.4. Experimental Results.....	144
5.2.4.1. Parameter Tuning.....	144
5.2.4.2. Algorithm Performance Analysis.....	146
5.3. A Noisy Genetic Algorithm for Stochastic, TimeV arying Networks.....	149

5.3.1. Sampling Fitness Function.....	151
5.3.2. Sampling Design.....	153
5.3.3. Constraint Handling.....	155
5.3.4. Illustrative Example.....	157
5.4. A Noisy Genetic Algorithm for Multicriteria, Stochastic, TimeV arying Networks.....	161
Chapter 6. Conclusions and Extensions.....	165
6.1. Synthesis.....	165
6.2. Future Extensions.....	169
Appendices.....	173
Appendix A Illustrative Example for The APS Algorithm.....	173
Appendix B Mathematical Formulation of the SEscape Algorithm.....	182
References.....	183

List of Tables

Table 2.1: Expected travel times and costs.....	23
Table 2.2: Notation.....	28
Table 2.3: Notation employed in the APS algorithm.....	37
Table 2.4: Notation employed in the ALEDS I and II algorithms.....	44
Table 2.5: Average run times in c.p.u. seconds for the APS algorithm on a 25 node network.....	54
Table 2.6: Average run times in c.p.u. seconds for the ALEDS I and II algorithms.....	55
Table 2.7: Average run times in c.p.u. seconds for the ALEDS II algorithm.....	57
Table 3.1: Probabilistic time and cost data.....	69
Table 3.2: Average run time comparisons.....	75
Table 4.1: Arc travel times and capacities for the example network.....	113
Table 4.2: The PTD residual network.....	113
Table 4.3: Backward arc information.....	114
Table 4.4: Experimental results.....	119
Table 5.1: Arc capacities and costs associated with each arc.....	141
Table 5.2: Results for network with 25 nodes and 60 time intervals.....	147
Table 5.3: Results for network with 100 nodes and 60 time intervals.....	148
Table 5.4: Results for the SEscape problem.....	149
Table 5.5: Fitness evaluation.....	157
Table 5.6: Random travel times and capacities.....	158
Table 5.7: Optimal values for 27 realizations.....	160

List of Figures

Figure 2.1: MSTV example network.....	21
Figure 2.2: Pareto-optimal hyperpath solutions.....	24
Figure 2.3: Illustration of Vector Labels at Node i	34
Figure 2.4: Example for the APS algorithm.....	35
Figure 3.1: MSTV example network.....	61
Figure 3.2: APPS solution.....	62
Figure 3.3: Illustrative example.....	68
Figure 3.4: Solutions for iteration 1.....	71
Figure 3.5: APPS solutions.....	72
Figure 3.6: APPS from node 1 to node 5.....	73
Figure 3.7: APPS/ELB run time comparisons.....	75
Figure 4.1: SEscape problem.....	77
Figure 4.2: Time-varying travel times and STV capacities.....	80
Figure 4.3: Example network.....	91
Figure 4.4: Stochastic arc capacities.....	94
Figure 4.5: The PTD residual network.....	99
Figure 4.6: The PTD backward arc.....	100
Figure 4.7: The PTD residual network after completion of $(n-2)$ unit shipments...	109
Figure 4.8: The PTD residual network after completion of $(n-1)$ unit shipments...	110
Figure 4.9: Example network.....	113
Figure 4.10: PTD residual network.....	115

Figure 4.11: Shipping one unit on path 1-3-2-4.....	116
Figure 4.12: Optimal solution.....	117
Figure 5.1: Solution representation.....	133
Figure 5.2: Genetic algorithm structure.....	139
Figure 5.3: Illustrative example.....	140
Figure 5.4: Arc attribute realization on arcs 1 and 2.....	156
Figure 5.5: STV capacitated network.....	158
Figure 5.6: Minimum time flow.....	159
Figure 5.7: Optimal pattern of flow.....	164

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION

This dissertation addresses two classes of network flow problems in networks with multiple, stochastic and time-varying attributes. The first problem class is concerned with providing routing instructions with the ability to make updated decisions as information about travel conditions is revealed for individual travelers in a transportation network. Such instructions are useful in a variety of applications, including selection of routes for hazardous materials transport, emergency response operations (medical, police, fire), intelligent transportation systems (ITS), and data networks. The second problem class is concerned with the determination of optimal *a priori* path flows for evacuation in capacitated networks, where the time-dependent and stochastic nature of arc attributes and capacities inherent in these problems is explicitly considered. Given arc capacity restrictions, a single path may not be able to accommodate all of the flow and it may be necessary to select a set of paths along which the flow will be shipped. A solution in this context, thus, consists of a set of paths and corresponding amount of flow to be shipped along each path.

In most of the published literature, the primary criterion used in determining an optimal path is either travel time or distance. However, in many real-world applications, other criteria may be of equal or greater importance. For example, in transportation applications, one may prefer a path that simultaneously minimizes travel time, distance, cost and accident likelihood. Identification of a single solution

that is best with respect to all criteria is often impossible. Rather, a set of Pareto-optimal (also referred to as efficient or non-dominated) solutions often exists. A solution is Pareto-optimal if and only if there are no other solutions that are better in at least one criterion and equal in the remaining criteria.

Numerous efficient algorithms exist for finding optimal solutions to network flow problems, where network attributes are given by time-invariant, deterministic quantities. However, often the attributes in a transportation network are stochastic and time-varying in nature. For example, arc travel times change over time due to time-of-day variations in traffic congestion. Furthermore, future travel times can at best be known *a priori* with uncertainty due to unforeseen events, such as poor roadway conditions, vehicle breakdowns, traffic accidents, and driver behavior. Uncertainty also exists as a result of measurement inaccuracies. Likewise, there are many applications for which the capacity of an arc may not be known with certainty and probability distribution functions or expected values may vary over time. In this dissertation, future arc capacities, travel times and other travel criteria are random variables with probability distribution functions that vary with time, i.e. multicriteria, stochastic and time-varying (MSTV) networks are considered. Explicit consideration of the dynamic and uncertain nature of multiple arc attributes in mathematical representations of real-world problems can significantly improve the utility and actual performance of the solutions.

In Sections 1.2 and 1.3, the specific problems that are addressed in this dissertation are introduced and discussed. A brief overview of the main contributions of this work is provided in Section 1.4.

1.2 PROBLEM CLASS I

A host of efficient algorithms exist in the literature for finding optimal paths in deterministic networks, where only a single criterion is considered. Few works have explicitly and simultaneously considered the dynamic and uncertain nature of multiple path attributes. In this dissertation, real-world complexities of path selection through explicitly considering the inherent variability in travel conditions, as well as the multiobjective nature of many path selection decisions are addressed.

In networks with stochastic, time-varying (STV) travel times, two problem classes may be considered: the *a priori* path and the time-adaptive path strategy problems. In both problem classes, solutions are obtained prior to the trip. The former results in a unique path that is defined in its entirety. The latter, in contrast, produces a set of path strategies that enables the traveler to select the best next direction from each intermediate location depending on the actual arrival time at that location. Such path strategies can be viewed as hyperpaths. In both problem classes, solutions are obtained prior to the trip. While solutions of both classes are provided *a priori*, in this dissertation the former is referred to as an *a priori* solution while the latter is referred to as a hyperpath solution. A solution approach that uses this time-adaptive feature may be desirable in providing real-time routing instructions. Similar problem classes can be defined in MSTV networks. An extension of the adaptive path problem with relation to MSTV networks is considered in this dissertation. The adaptive path strategy in MSTV networks is referred to as an adaptive Pareto-optimal path strategy (i.e. Pareto-optimal hyperpaths). Three specialized label-correcting algorithms are presented for identifying all or a subset of the adaptive Pareto-optimal solutions with

respect to the expected value of each criterion from each node to a desired destination for each departure time in the period of interest. A brief description of these exact solution techniques is given next. The detailed conceptual framework and specific computational steps for solving these problems are presented in Chapters 2 and 3.

Adaptive Pareto-Optimal Strategy (APS) Algorithm

The Adaptive Pareto-Optimal Strategy (APS) algorithm generates all adaptive Pareto-optimal path strategies in MSTV networks. The concepts of Pareto-optimality and time-adaptive strategy are combined to generate the set of Pareto-optimal path strategies that enable the traveler to select a direction among all Pareto-optimal solutions at each node in response to knowledge of the arrival time at the intermediate nodes. Solution paths that seek to minimize the expected value of multiple criteria are sought from all origins to a specified destination for all departure times in a period of interest.

The APS algorithm is a specialized label-correcting algorithm for use with multiple criteria. The algorithm proceeds in an iterative manner, working backward starting from the destination node. One or more hyperpaths may be generated from each node at each departure time. That is, hyperpaths are constructed through each of the Pareto-optimal subhyperpaths at a successor node. Upon termination, all Pareto-optimal hyperpaths with respect to the expected value of the considered criteria from each origin to the destination node for each departure time are generated.

Adaptive Least Expected Disutility Strategy (ALEDS I & II) Algorithm

The generation of all Pareto-optimal hyperpaths may require enormous computational effort; thus, the ALEDS I and II algorithms are proposed to provide the single “best compromise” solutions by explicitly representing the decision maker’s preference structure through a disutility function. Rather than generate all Pareto-optimal hyperpaths and *a posteriori* select a single solution, the ALEDS algorithm relies on the use of a preference function in the form of a linear utility function to produce only a single hyperpath solution, i.e. the one that minimizes the expected disutility.

The ALEDS I algorithm works by computing the expected value for each criterion prior to determining the disutility for the associated hyperpath. To efficiently generate the least expected disutility (LED) hyperpaths, an enhancement to the ALEDS I algorithm is presented, referred to as the ALEDS II algorithm. Unlike the ALEDS I algorithm that requires computation and storage of the expected value for each criterion, this second variation assesses the expected disutility directly and keeps only the minimum value for each node and departure time. Both algorithms terminate with the LED hyperpaths from all nodes to a specified destination for all departure times in the period of interest. While the ALEDS I algorithm is more intuitive, the ALEDS II algorithm provides substantial improvements in computational complexity and storage requirements.

Adjustable Preference Path Strategy (APPS) Algorithm

The Adjustable Preference Path Strategy (APPS) algorithm generates adaptive path strategies that seek to minimize the expected value of each of multiple criteria from

all origins to a specified destination for all departure times in a period of interest. These solution strategies allow a traveler to change his or her preference for the criterion upon which path decisions should be made at intermediate locations en route to the destination and then adaptively select the best path with respect to the expected value of the chosen criterion at each node in response to knowledge of experienced travel times on the arcs. Such adaptive strategies are referred to herein as adjustable preference path strategies (APPS). The APPS algorithm is proposed to determine such adaptive strategies in MSTV networks. The APPS algorithm is particularly useful for providing real-time path finding assistance in traffic networks.

The APPS algorithm determines the APPS by checking at each departure time if using the hyperpaths associated with the node along the hyperpaths from its predecessor nodes generates a lower expected value of one or more of the criteria from these predecessor nodes to the destination than previously considered hyperpaths.

1.3 PROBLEM CLASS II

The second problem class considered in this dissertation is concerned with the generation of the optimal set of *a priori* path flows in networks, where arc capacity restrictions are considered. While network flow problems in this context have wide applicability in many different arenas, this dissertation focuses on the development of evacuation plans for emergency escape from a large building. Most of the related works proposed in the literature for determining exact solutions focus on static and deterministic problems. In light of the intelligent evacuation, rescue and recovery

(IERR) concept (Miller-Hooks and Krauthammer, 2002), real-time assessment of the extent of blast damage to a building's structures makes it possible to derive probabilistic passageway traversal times and capacities over time. In this dissertation, the uncertain and time-varying nature of arc capacities inherent in emergency situations is explicitly considered. Stochastic and time-varying (STV) arc capacities impede the effectiveness of implementing the solution obtained from conventional deterministic approaches, because there might be some probability that the capacity of an arc cannot accommodate all the flow attempting to traverse it. This motivates consideration of a performance measure that takes into account these probabilities in evaluating solution path flows.

A brief discussion of the capacitated network flow problems addressed in this dissertation is given next. The detailed conceptual framework and specific computational steps for solving the problems are presented in Chapters 4 and 5.

The Safest Escape Problem

The Safest Escape (SEscape) problem is to determine optimal path flows in dynamic (i.e. flow moves through the network over time and arc capacities are recaptured over time) networks, where arc travel times are time-varying and arc capacities are random variables with probability distribution functions that vary with time. Traditionally, the evacuation time, i.e. the time until the last evacuee exits the disaster area, is considered in developing evacuation plans. However, when capacities of passageway can be at best known only probabilistically, it may be beneficial to route people to a longer time path with high likelihood of successful arrival at the safe area than to a

faster path with small likelihood of successful arrival. For this reason, the probability of successful escape is explicitly considered herein.

An exact algorithm, the SEscape algorithm, is proposed to determine the pattern of flow that maximizes the minimum path probability of successful arrival of supply at the sink. In life-threatening situations, it is important to avoid routing any evacuee to a path that would have a high likelihood of failing by the time the evacuee arrives at that location. The SEscape algorithm extends the Time-Dependent Quickest Flow Problem (TDQFP) algorithm of Miller, Hooks and Stock Patterson (2004) for solving the TDQFP in deterministic, time-varying networks for use in stochastic environments. That is, it iteratively determines the maximum probability paths from source to sink in a residual network and incrementally pushes flow along the paths until all demand is fulfilled. The SEscape algorithm terminates with a set of paths from the source to the sink and the corresponding number of units to be shipped along each path such that the minimum probability of arrival at the sink is maximized. Through the implementation of the technique given in Miller, Hooks and Stock Patterson (2004) for efficiently converting multi-source, multi-sink network flow problems to single source, single sink problems, the SEscape algorithm can be used to solve the SEscape problem given multiple sources and multiple sinks. In the context of emergency evacuation, such solutions minimize the risk incurred by the people who are forced to take the greatest risk.

The rationale and the design of the specific computational steps for addressing the SEscape problem are provided in Chapter 4.

Minimum Cost Network Flow Problems in MSTV, Capacitated Networks

While the SEscape problem assumes STV capacities, arc travel times are treated as deterministic quantities. In many situations, however, future arc travel times may not be known *a priori* with certainty. Difficulty arises when assessing exact solutions of network flow problems with STV arc travel times and capacities, i.e. in STV capacitated networks. If both quantities are modeled as discrete random variables, a particular combination of possible travel times (and other attributes) and capacities at each discrete point in time results in a realization of such a STV network. That is, the network can take on a number of discrete states and this number grows exponentially with the size of the network and number of possible travel times and capacities along each arc for each departure time. The optimal solution for one state may not be feasible for another state and it is possible that no feasible solution exists for any network realization. Exact solution to problems of this nature that rely on enumeration of all states will require substantial computational effort. A methodological framework that can provide competitive approximate solutions with reasonable computational effort is proposed. Specifically, a meta-heuristic based on the principles of noisy genetic algorithms (NGAs) is presented for determining optimal path flows in dynamic networks, where arc traversal times and capacities are random variables with probability mass functions that vary with time.

A genetic algorithm (GA) is first presented for solving the problem of determining the optimal flow pattern, where the arc travel times are assumed to be deterministic and time-varying. Specifically, the solution approach seeks the minimum cost flow for shipping a given amount of supply. The performance of the

GA is compared with that of the exact techniques, specifically a no-waiting version of the TDQFP algorithm (Miller-Hooks and Stock Patterson, 2004), where the arc capacities are deterministic and time-varying, and the SEscape algorithm (Chapter 4), where the arc capacities are known only with uncertainty.

In the GA, the solution representation structure is specifically designed to accommodate only feasible solutions. Each chromosome contains several genes that form a pattern of flow. Each gene consists of two parts. The first part contains a sequence of arcs forming a path from the source to the sink. The second part indicates the number of flow units to be sent through the path. Only feasible solutions are generated in the initial population and solution feasibility is maintained at intermediate stages of the algorithm through the application of specially designed operators, including crossover and mutation operators. In each generation, binary tournament selection is employed to select solutions to enter the next generation. The GA is extended to address the problem in more difficult, STV and MSTV capacitated networks, where no exact algorithms exist.

To address the problem of finding optimal path flows in STV and MSTV capacitated networks, noisy genetic algorithms (NGAs) are implemented. A sampling fitness function is used to evaluate solutions in each iteration. Unlike in the application of the GA to deterministic problems, infeasible solutions are permitted, but a penalty is incurred for violating the problem constraints. In many applications that can be modeled as network flow problems, multiple conflicting objectives are involved. For example, a set of paths that maximize the expected flow and simultaneously minimize total time may be desired in building evacuation. Such

objectives may be conflicting in nature. Therefore, extension of the NGA for use in addressing multicriteria dynamic network flow problems with stochastic, time-varying arc attributes, including arc capacities, is presented.

Details of the GAs and NGAs proposed here for addressing single objective and multiobjective problems related to evacuation are given in Chapter 5.

1.4 CONTRIBUTIONS

The main contributions of this dissertation are as follows.

Problem class I

I.1) The development of an exact technique for generating all adaptive Pareto-optimal solutions of a multicriteria optimal path problem in stochastic, time-varying networks.

I.2) The design of the specific procedural steps for directly determining a single “best” hyperpath in MSTV networks by explicitly representing the decision maker’s preference structure through a disutility function.

I.3) The rationale and the design of the specific computational steps for determining adjustable preference path strategies (APPS) that permit a traveler to adapt his or her path according to both revealed travel conditions and the traveler’s changing preferences at intermediate locations while traveling to the destination in MSTV networks.

Problem class II

II.1) The development of the conceptual framework and exact algorithm for determining emergency evacuation strategies in dynamic, capacitated networks,

where the risk incurred by the person or people who are forced to take the greatest risk is minimized.

II.2) The development of a meta-heuristic for addressing the problem of determining optimal path flows in dynamic networks, where multiple arc attributes and capacities are stochastic and time-varying.

CHAPTER 2

ADAPTIVE PARETO-OPTIMAL PATH STRATEGIES

2.1 INTRODUCTION

This chapter addresses the problem of determining adaptive path strategies in stochastic, time-varying (STV) networks with multiple arc attributes, i.e. in multicriteria STV, or MSTV, networks. In MSTV networks, multiple arc attributes are associated with each arc, each of which is a random variable whose probability distribution function (PDF) varies with time. With multiple criteria, it is unlikely that there exists a single path between a given origin-destination pair that is best with respect to all criteria. Instead, the solution of a multicriteria “optimal” path problem will be a set of Pareto-optimal (or non-dominated) paths.

Let $P_a = \{P_a^1, P_a^2, \dots, P_a^r\}$, where r is the number of criteria under consideration and P_a^k , $k \in \{1, 2, \dots, r\}$, is the value with respect to criterion k for path a in a deterministic, time-invariant network. Then,

P_a is non-dominated if no b ($\neq a$) exists between the same origin-destination pair

such that $P_b^k \leq P_a^k$ for all $k \in \{1, 2, \dots, r\}$ and $P_b^h < P_a^h$ for some $h \in \{1, 2, \dots, r\}$

(condition 1).

In this chapter, exact algorithms are proposed for addressing adaptive path problems, where arc attributes are stochastic and time-varying. Adaptive paths comprise a set of path strategies that enable the traveler to select a direction among all Pareto-optimal solutions at each node in response to knowledge of the arrival time at

the intermediate nodes. Such paths can be viewed as hyperpaths and are referred to in this way herein. The first algorithm generates all adaptive Pareto-optimal path strategies (referred to as Pareto-optimal hyperpaths) in MSTV networks. Specifically, solution paths that seek to minimize the expected value of multiple criteria are sought from all origins to a specified destination for all departure times in a period of interest.

Regardless of the technique employed or the application considered, generation of all Pareto-optimal paths may require generation of all possible paths, because all paths may be Pareto-optimal. Any technique that generates all Pareto-optimal solutions has exponential worst-case computational complexity. Therefore, two computationally efficient variations of an additional algorithm are proposed that rely on the use of a preference function in the form of a linear utility function to produce only a single hyperpath, i.e. the one that minimizes the expected disutility. These techniques address real-world complexities of path selection through explicitly considering the inherent variability in travel conditions, as well as the multiobjective nature of many path selection decisions. Moreover, they take advantage of the traveler's ability to make updated decisions as information about uncertain quantities is revealed.

Section 2.2 provides a brief discussion of the works that have been proposed in the literature for addressing a variety of related optimal path problems.

2.2 LITERATURE REVIEW

2.2.1 Single Criterion Shortest Path Problems

A great number of researchers have been studying a variety of optimal path problems. These efforts to solve this problem have ranged in focus from solutions in the simplest static networks to very complicated stochastic, and/or multicriteria networks. Most of these efforts concentrate on the determination of the shortest path in deterministic networks with travel time as a single criterion. Few take into account the stochastic nature of the network elements, where issues of random variables and associated probability functions are addressed.

The simplest one is the classical shortest path problem in static networks with a single arc attribute. There are two general approaches used to solve the shortest path problem: label-setting and label-correcting algorithms (see Ahuja, Magnanti and Orlin (1993) for more detail). Both algorithms initially establish a temporary distance label to each node from a given origin that maintains an upper bound on the shortest path distance from the origin to that node. The labels are updated iteratively. In the label-setting algorithm, a label is selected and made permanent (i.e. represents the final shortest distance from the origin to the considered node) at each iteration. By contrast, in the label-correcting algorithm, all labels are assumed to be temporary and will become permanent only when the algorithm terminates. Note that neither algorithm can solve for the shortest path in networks containing negative cycles since they could result in the incorrect shortest path.

Many approaches have been proposed in the literature for finding the shortest path in networks with time-varying but deterministic arc travel times (Cooke and

Halsey, 1966; Dreyfus, 1969; Orda and Rom, 1990; Ziliaskopoulos and Mahmassani, 1993; Kaufman and Smith, 1993; and Chabini, 1998). Of interest is the work of Ziliaskopoulos and Mahmassani (1993). They introduced the Time-Dependent Least Time Problem (TDLTP) algorithm for finding the shortest paths in discrete time-varying networks for all discrete departure times. The TDLTP algorithm is a label-correcting based algorithm where no waiting is permitted at any node. The arc travel times are non-negative real values. After the period of interest, the arc travel times are fixed and are equal to those of the last time interval. Because of prohibited waiting, the TDLTP algorithm does not deal with optimal departure times at the nodes and cannot determine the shorter path which may occur if delay in departure through waiting at intermediate nodes is allowed and the network is non-FIFO.

Hall (1986), Miller-Hooks and Mahmassani (2000), and Pretolani (2000) studied variations of this problem in stochastic, time-varying (STV) networks. Each of these works addressed two problem classes. The first seeks an *a priori* solution and the second seeks time-adaptive path strategies. A solution in the former problem class is a unique path, which is chosen entirely before starting the trip and is fixed for each departure time. Like general shortest path algorithms, the *a priori* solution provides a single best path for a whole trip at a particular departure time from the origin. Fu and Rilett (1998) proposed a heuristic for the *a priori* path problem. Alternatively, if the traveler is permitted to adjust the path at each node in accordance with known arrival times and trip information experienced at previously visited nodes, a more preferable path (e.g. shorter expected travel time) can be found, referred to as the time-adaptive path strategies. By this time-adaptive travel decision, there no longer exists a single

best path. The single best path cannot be revealed until the trip is completed since the arrival time at each node cannot be known before travel. Such time-adaptive path strategies are referred to as hyperpaths (Miller-Hooks, 2000), where each path segment depends on arrival time information gained as travel is completed. Other works that address adaptive path problems in stochastic networks include Polychronopoulos and Tsitsiklis (1996), Waller and Ziliaskopoulos (2003), Cheung (1998), Fu (2001), and Provan (2003). The first two of these works account for arc cost dependencies.

2.2.2 Multicriteria Optimal Path Problems

Numerous works propose solution procedures for multicriteria path problems, where all arc attributes are assumed to be deterministically known and time-invariant. Climaco and Martins (1982) proposed an algorithm based on a K -shortest paths concept for solving bicriterion shortest path problems. Martins (1984) developed two algorithms for generating all Pareto-optimal paths. One algorithm is a generalization of Hansen's label-setting approach to this problem (Hansen, 1980). Corley and Moon (1985) developed a label correcting-based algorithm for generating all Pareto-optimal paths. Zografos and Davis (1989) employed goal programming for routing hazardous materials in multiobjective static networks. In the context of traffic assignment, Dial (1979) proposed a technique for generating combined route-mode choices that are Pareto-optimal. Other related works include Warburton (1987), Stewart and White (1989), Mote et al. (1991), Murthy and Her (1992), and Murthy and Olson (1994).

Because any technique that generates all Pareto-optimal solutions has exponential worst-case computational complexity, some researchers have applied utility functions to address multidimensional optimal path problems in static networks. Modesti and Sciomachen (1998) used Dijkstra's algorithm to determine paths that minimize linear utility functions in multimodal deterministic networks in solving a multiobjective traffic assignment problem. Loui (1983) and Eiger et al. (1985) showed that when the utility function is linear or exponential, any labeling-based algorithm can be used to find the optimal path in static networks without violating Bellman's principle. Carraway et al. (1990) proposed a generalization of dynamic programming based upon the weak principle of optimality for use with non-monotonic utility functions. Their generalized dynamic programming approach addresses the multicriteria shortest path problem in acyclic, static networks when Bellman's principle may be violated. Henig (1985) proposed approaches to find the bicriterion shortest path when the utility function is quasiconcave or quasiconvex. Mirchandani and Wiecek (1993) reduced the stochastic shortest path problem to a multiattribute optimal path problem with a nonlinear monotonic utility function. For the case of a convex utility function, a line-search approach was proposed to solve cases with two arc attributes. Mirchandani and Soroush (1985) developed an efficient algorithm to solve the problem of finding an optimal path in stochastic networks with a quadratic utility function. To solve the same problem, Murthy and Sarkar (1996) proposed a label-setting based approach that embeds a relaxation-based pruning technique.

Several works investigate multicriteria path problems in stochastic, time-invariant networks. Turnquist (1987) suggested the use of simulation together with a labeling-based path algorithm to address this problem. Specifically, for each replication a realization of each arc attribute is randomly generated and the set of Pareto-optimal paths between an origin-destination pair is determined. This is repeated over multiple replications and the frequency with which the paths arise as Pareto-optimal is used to estimate the probability that each path will be Pareto-optimal. Given normally distributed arc attributes, Wijeratne et al. (1993) developed the Stochastic Multiobjective Shortest Path algorithm for finding a set of paths in stochastic, time-invariant networks. They presented an approximation to stochastic dominance to compare path distribution functions for a single stochastic criterion. The problem is extended to multiple criteria, but the criteria are reduced to two deterministic factors; and hence, the final problem is reduced to a deterministic, multiobjective problem.

All of the previously discussed works assume that the arc attributes are time-invariant. However, there are many applications for which multiple attributes, such as travel time, travel cost, population exposure, and incident rate, may be time-varying. Nozick et al. (1997) developed an integrated routing and scheduling approach for solving a multicriteria problem related to hazardous materials shipments with time-varying, but deterministic, attributes. The resulting solutions are a set of route-departure time combinations. This approach cannot guarantee that all Pareto-optimal paths will be generated. In STV networks, Miller-Hooks and Mahmassani (1998b) provided label correcting-based methodologies to generate all *a priori* Pareto-optimal

paths in STV networks with respect to several dominance definitions. Chang et al. (forthcoming) presented a heuristic for solving an *a priori* multicriteria path problem in STV networks, where all arc attributes are assumed to be continuous random variables. Nielsen et al. (2003) studied the problem of finding shortest hyperpaths in STV networks with two criteria. Three classes of problem were considered: 1) minimize the expected travel time and cost; 2) minimize the maximum travel time and cost; and 3) minimize the expected travel time and maximum cost. The two-phase method originally designed for solving the bicriterion shortest path problem was modified for solving the bicriterion hyperpath counterpart. The proposed approach requires the construction of a time-expanded hypergraph. One can notice that the capability of the two-phase method is bounded on shortest path problems with two criteria. It appears that there are no published works that consider the generation of all adaptive Pareto-optimal strategies in multicriteria, stochastic and time-varying (MSTV) networks.

In the next section, the solution nature of the Pareto-optimal hyperpaths in MSTV networks is illustrated through an example problem and properties are developed. In Section 2.4, network notation and problem definitions are given. In Section 2.5, exact algorithms are proposed to address path problems in MSTV networks. This is followed by notes on algorithm implementation in Section 2.6. Results of computational experiments designed to examine the average computational performance of the proposed algorithms are presented in Section 2.7. Finally, discussion and conclusions are given in Section 2.8.

2.3 PARETO-OPTIMAL HYPERPATHS IN STV NETWORKS

In this section, an example is given to illustrate the nature of hyperpath solutions in MSTV networks. The example is also used to show that, similar to path problems in STV networks with a single attribute associated with each arc, when multiple STV attributes exist, one can also make improved decisions by adaptively choosing the path. In Figure 2.1, a MSTV network with two arc attributes, i.e. travel time (criterion I) and cost (criterion II), is shown. It is assumed that both time and cost are time-varying and are known only probabilistically. Waiting is not permitted at any node and the arc attributes are assumed to be independent over space and time and independent of each other.

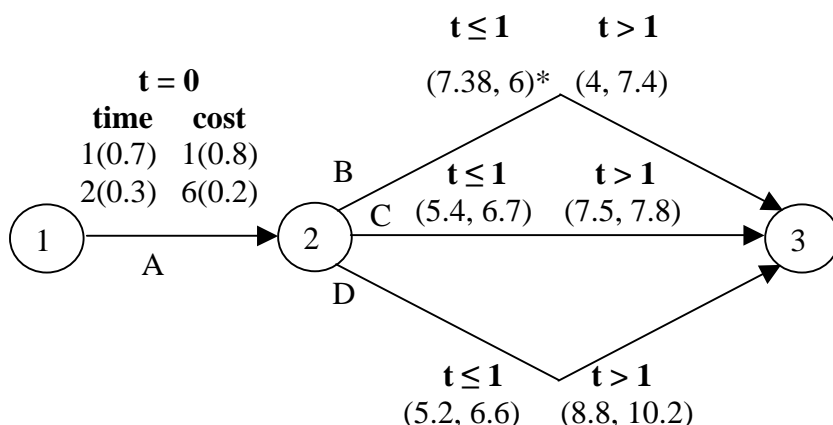


Figure 2.1. MSTV example network.

In Figure 2.1, the arc attributes are expressed either as expected values (for subpaths B, C and D) or as probability mass functions where the probability of each possible outcome is given parenthetically (for arc A). For example, there are two possible travel times on arc A when departing from node 1 at $t = 0$: 1 with probability 0.7 and 2 with probability 0.3. There are also two possible costs: 1 with probability

0.8 and 6 with probability 0.2. For the sake of simplicity, the expected values are directly given for each attribute of subpaths B, C, and D, e.g. the expected travel time and cost for path B at time $t \leq 1$ are 7.38 and 6, respectively. It can be seen that three paths, paths A-B, A-C and A-D, exist between node 1 (origin) and node 3 (destination). Suppose the traveler departs from node 1 at $t = 0$. The expected path attributes for these three paths can be found as follows (see Miller-Hooks and Mahmassani (2000) for additional detail on these computations for both *a priori* and adaptive path problems with a single arc attribute).

P_1) Path A-B

Expected travel time: $(1+7.38) \cdot 0.7 + (2+4) \cdot 0.3 = \mathbf{7.67}$.

Expected travel cost: $(1+6) \cdot 0.7 \cdot 0.8 + (6+6) \cdot 0.7 \cdot 0.2 +$

$$(1+7.4) \cdot 0.3 \cdot 0.8 + (6+7.4) \cdot 0.3 \cdot 0.2 = \mathbf{8.42}.$$

Employing similar computations to determine the expected values for paths A-C and A-D, we find the expected travel time and cost of each path to be (7.33, 9.03) and (7.58, 9.68), respectively. As discussed in more detail in Section 2.4, by extending condition (1) for use in STV networks, dominance at a particular departure time can be established by means of pairwise comparisons of expected values (for either the *a priori* or adaptive problem). Thus, for departure time $t = 0$, two *a priori* Pareto-optimal paths exist for this example problem: paths A-B and A-C. That is, path A-D is dominated by path A-C.

In the adaptive version of this multicriteria path problem, the traveler can postpone his/her choice between subpaths B, C and D until arrival at node 2. The expected time for one such adaptive solution is computed as follows.

H₁) Path A-B if arrival at node 2 is $t = 1$ or path A-C if arrival at node 2 is $t = 2$

Expected travel time: $(1+7.38) \cdot 0.7 + (2+7.5) \cdot 0.3 = 8.72$.

Expected travel cost: $(1+6) \cdot 0.7 \cdot 0.8 + (6+6) \cdot 0.7 \cdot 0.2 +$

$$(1+7.8) \cdot 0.3 \cdot 0.8 + (6+7.8) \cdot 0.3 \cdot 0.2 = 8.54.$$

Similar computations were employed to determine the expected travel times and costs for each possible hyperpath for this problem. The expected travel times and costs for all hyperpaths are provided in Table 2.1. Only H₄ and H₇ are non-dominated. These hyperpath strategies are portrayed in Figure 2.2. Both Pareto-optimal solutions instruct the traveler to follow arc A at $t = 0$. Since waiting is not permitted, both solutions indicate that the next move from node 2 is subpath B for arrival time 2. For arrival time 1, one can choose either subpath D by H₄ or subpath B by H₇. Note that, for this example, Pareto-optimal path A-C to the *a priori* problem is dominated by H₄. We establish a number of relationships between *a priori* and adaptive Pareto-optimal paths for MSTV networks in Propositions 2.1 through 2.5.

Table 2.1. Expected travel times and costs.

Hyperpath index	Resulting strategy by arrival time at node 2		(Expected time, Expected cost)
	t = 1	t = 2	
H ₁	B	C	(8.72, 8.54)
H ₂	C	B	(6.28, 8.91)
H ₃	B	D	(9.11, 9.26)
H ₄	D	B	(6.14, 8.84)
H ₅	C	D	(7.72, 9.75)
H ₆	D	C	(7.19, 8.96)
H ₇	B	B	(7.67, 8.42)
H ₈	C	C	(7.33, 9.03)
H ₉	D	D	(7.58, 9.68)

Note: Hyperpaths H₇, H₈ and H₉ are identical to *a priori* solutions A-B, A-C and A-D, respectively.

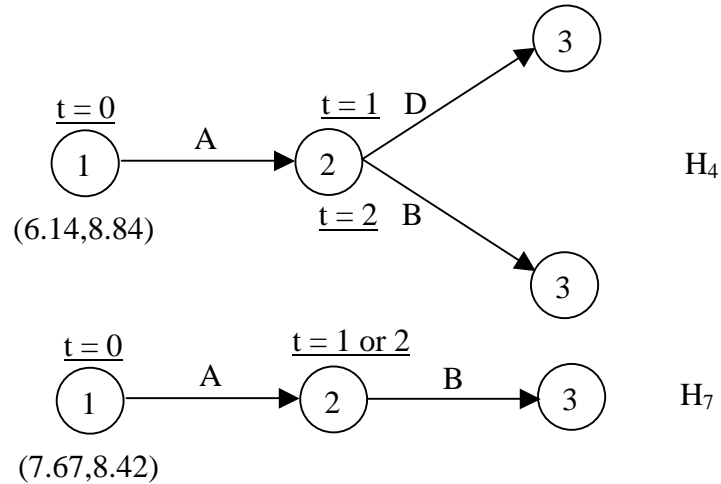


Figure 2.2. Pareto-optimal hyperpaths.

Proposition 2.1. Any Pareto-optimal solution to the adaptive path problem cannot be dominated by any *a priori* solution.

Proof. Assume an *a priori* path P exists that dominates a Pareto-optimal hyperpath H_1 . If such an *a priori* path is not dominated by any other Pareto-optimal hyperpath, this *a priori* path must serve as a Pareto-optimal hyperpath, and thus, will trivially fulfill this statement. If, on the other hand, *a priori* solution P is dominated by another Pareto-optimal hyperpath, H_2 , hyperpath H_1 would be dominated by H_2 , contradicting our assumption that H_1 is Pareto-optimal. ♦

In the single criterion adaptive LET path problem for a given departure time from a given origin, the expected time of the solution hyperpath provides a lower bound on the expected time of the *a priori* LET path (proof of this is given in Miller-Hooks and Mahmassani, 2000). In Proposition 2.2, we show that this concept is not necessarily true when multiple criteria are considered.

Proposition 2.2. A Pareto-optimal solution to the adaptive path problem may not contain even one criterion for which its expected value is less than or equal to that of all *a priori* solutions.

Proof. Assume a Pareto-optimal hyperpath must contain at least one criterion for which its expected value is less than or equal to that of all *a priori* solutions. To establish a counter example, the example network shown in Figure 2.1 is employed with one adjustment in arc travel time, i.e. the expected travel time on subpath C for $t > 1$ is changed from 7.5 to 3.95. With this adjustment, the three *a priori* solutions have expected values for each of the two criteria as given parenthetically: P_1 (7.67, 8.42), P_2 (6.27, 9.03) and P_3 (7.58, 9.68). For the same example, four Pareto-optimal hyperpaths can be identified: H_1 (7.65, 8.54), H_4 (6.14, 8.84), H_6 (6.13, 8.96) and H_7 (7.67, 8.42). Hyperpath H_1 is Pareto-optimal and does not contain a criterion for which its expected value is less than or equal to that of all *a priori* solutions, contradicting the assumption, and thus, establishing a counter example. ♦

The counter example established in the proof of Proposition 2.2 leads to another concept for establishing a bound on the expected values for each criterion for the *a priori* problem.

Proposition 2.3. The lowest expected value of all Pareto-optimal solutions to the adaptive path problem for each criterion is less than or equal to that of any *a priori* solution.

Proof. For each criterion, if an *a priori* solution exists such that its expected value on this criterion is the lowest of all Pareto-optimal hyperpaths, this *a priori* solution

would be Pareto-optimal to the adaptive problem (i.e. would serve as a Pareto-optimal hyperpath), thus, providing the lowest value for that criterion. ♦

Another relationship that can be extended from the single criterion problem (Miller-Hooks and Mahmassani, 2003) is given in Proposition 2.4.

Proposition 2.4. A Pareto-optimal path to the *a priori* problem may not contribute to any Pareto-optimal solution to the adaptive path problem.

Discussion. The example network shown in Figure 2.1 provides a counter example. Although *a priori* solution A-C is non-dominated for the *a priori* problem, when adaptive decisions can be made, it is never best to continue from node 2 along subpath C since all adaptive solutions containing subpath C are dominated.

Proposition 2.5. A dominated path to the *a priori* problem may contribute to a Pareto-optimal solution to the adaptive path problem.

Discussion. Again, the example network in Figure 2.1 provides a counter example. Path A-D is dominated for the *a priori* problem. However, when adaptive solutions are permitted, path A-D will be a Pareto-optimal strategy for a given departure time from node 2, thus, contributing to a Pareto-optimal solution to the adaptive path problem.

The next section provides the network notation along with problem definitions for the two problem classes addressed in this chapter.

2.4 NETWORK NOTATION AND PROBLEM DEFINITIONS

Similar notation for describing the network as used by Miller-Hooks and Mahmassani (2000) is employed herein. Let $G = (\mathcal{V}, \mathcal{A}, S, C, \mathcal{P}, R)$ be a finite digraph, where \mathcal{V} is the set of nodes and \mathcal{A} is the set of directed arcs connecting the nodes. $\Gamma^{-1}(i)$ denotes the set of predecessor nodes of node i , i.e. all $j / (j, i) \in \mathcal{A}$. Likewise, $\Gamma^{+1}(i)$ denotes the set of successor nodes of node i , i.e. all $j / (i, j) \in \mathcal{A}$. The period of interest, referred to as the peak period, is discretized into small time intervals represented by $S = \{t_0 + s\Delta t\}_{s = \{0, 1, 2, \dots, I\}}$, where Δt is the length of each interval of time. The peak period starts at time t_0 and ends at time $t_0 + I\Delta t$. Arc attributes are assumed to vary with time during this period. After this period, it is assumed that the arc attributes are stationary, taking the same values as at the last time interval, $t_0 + I\Delta t$. Multiple attributes are associated with each arc. The arc attributes are assumed to be discrete random variables with probability mass functions (PMFs) given by the sets (C, \mathcal{P}) (the set of arc attributes, $\{C^1, C^2, \dots, C^r\}$), and corresponding probabilities of occurrence, $\{\mathcal{P}^1, \mathcal{P}^2, \dots, \mathcal{P}^r\}$, where set $R = \{1, 2, \dots, r\}$ denotes the considered criteria.

For each arc $(i, j) \in \mathcal{A}$, $k \in R$, $C^k = \{c_{ij}^{kz_k}(t)\}_{z_k = 1, \dots, D}$ denotes the set of D possible arc values for criterion k for traversing the arc at departure time t . For each z_k possible arc value, $c_{ij}^{kz_k}(t)$ is assumed to be non-negative, real-valued with associated probability of occurrence, $\rho_{ij}^{kz_k}(t) \in \mathcal{P}^k$. In context of path finding assistance, travel time will often be a criterion that is considered. In such instances, we assume that travel time is the first of r criteria, and thus, throughout this chapter, $k = 1$ refers to the travel time criterion. Arc attribute values and corresponding

occurrence probabilities are specified upon entrance to an arc and are assumed to be static for that particular traveler until exiting the arc. This is sometimes referred to as the frozen link property (Orda and Rom, 1990). For each departure time occurring after the peak period, $t > t_0 + I\Delta t$, $c_{ij}^{kz_k}(t) = c_{ij}^{kz_k}(t_0 + I\Delta t)$ and $\rho_{ij}^{kz_k}(t) = \rho_{ij}^{kz_k}(t_0 + I\Delta t)$ $\forall k, z_k$ and (i,j) . The arc attributes are assumed to be independent over space and time and independent of each other. Table 2.2 summarizes this notation.

Table 2.2. Notation.

	Notation
\mathcal{V} :	set of nodes
\mathcal{A} :	set of directed arcs
S :	set of discrete time intervals
R :	set of criteria
D :	number of possible arc values for any criterion
r :	number of criteria
C^k :	set of discrete random variables for criterion k
\mathcal{P}^k :	set of occurrence probabilities associated with C^k
$c_{ij}^{kz_k}(t)$:	z_k - th arc value of criterion $k \in \{1,2,\dots,r\}$ for traversing arc (i,j) at departure time t
$\rho_{ij}^{kz_k}(t)$:	probability of occurrence associated with $c_{ij}^{kz_k}(t)$

For single criterion shortest path problems with time-varying travel times, a FIFO (First-In, First-Out) network requires that any vehicle departing from a particular node earlier than another vehicle must arrive at the next node before this other vehicle if they traverse the same path. Let $c_{ij}^k(t)$ be the arc weight on arc (i,j) at departure time t with respect to criterion k . As described in Kaufman and Smith (1993) for deterministic, time-varying networks, this FIFO condition (referred to as

the consistency assumption) can be stated as follows (recall, $k=1$, refers to the travel time criterion).

$$\text{For any arc } (i,j) \in \mathcal{A}, s + c_{ij}^1(s) \leq t + c_{ij}^1(t) \forall s, t \in \mathcal{S} \text{ and } s \leq t.$$

In the literature, FIFO conditions are defined with respect to travel time. However, in this work, we consider such conditions with respect to other criteria. Specifically, we establish similar conditions for non-travel time criteria for cases where a cost in terms of that criterion is or is not incurred as a consequence of waiting. The key to establishing these conditions, and the reason this is pertinent is that one can never do better to wait at any intermediate location in FIFO networks. With this in mind, the general form of the FIFO condition for any non-travel time criterion k can be stated as follows.

$$\text{For any arc } (i,j) \in \mathcal{A}, c_{ij}^k(s) \leq \omega^k(t-s) + c_{ij}^k(t) \forall s, t \in \mathcal{S}, s \leq t \text{ and } k(\neq 1) \in \mathcal{R},$$

where $\omega^k(t-s) \geq 0$ is a cost with respect to criterion k associated with waiting from time t to time s .

Miller-Hooks and Mahmassani (1998a) extended the consistency assumption of Kaufman and Smith to depict the FIFO condition with respect to travel time in STV networks.

$$\text{For any arc } (i,j) \in \mathcal{A}, \Pr\{s + c_{ij}^1(s) \leq t + c_{ij}^1(t)\} = 1 \forall s, t \in \mathcal{S} \text{ and } s \leq t,$$

where $c_{ij}^1(\cdot)$ is any possible travel time in a STV network.

We extend this FIFO condition to other criteria. For criterion k other than travel time, the FIFO condition in STV networks can be restated as follows.

For any arc $(i,j) \in \mathcal{A}$, $\Pr\{c_{ij}^k(s) \leq \omega^k(t-s) + c_{ij}^k(t)\} = 1 \quad \forall s, t \in \mathcal{S}, s \leq t$ and $k(\neq 1) \in \mathcal{R}$,

where $c_{ij}^k(\cdot)$ is any possible value for criterion k in a MSTV network.

For many applications, the FIFO condition may be violated with respect to one or more criteria. For example, one vehicle may overtake another vehicle or the cost of a train ticket may be significantly reduced at specific times of day. Such networks where FIFO conditions may be, but are not necessarily, violated are referred to as non-FIFO networks. In this work, we address this more general non-FIFO problem. While waiting in non-FIFO networks may lead to more favorable conditions, we assume no waiting is permitted at any intermediate location. The problem with no waiting is considered to be more difficult to solve than one that permits waiting (e.g. Chabini, 1998). Further, there are many applications, such as providing routing instructions to drivers in a traffic network, where waiting at intermediate locations is not an option.

Two problems are addressed in this chapter. The first problem seeks all Pareto-optimal hyperpath strategies with respect to the expected value of each criterion from each node to the destination for each departure time $t \in \mathcal{S}$. Let $H(t)$ be the set of all possible hyperpaths connecting an origin-destination pair for departure time t and let $\tilde{\theta}_a^k(t)$ be the random variable for the k^{th} criterion along a hyperpath $a \in H(t)$. $E[\tilde{\theta}_a^k(t)] = \{E[\tilde{\theta}_a^1(t)], E[\tilde{\theta}_a^2(t)], \dots, E[\tilde{\theta}_a^k(t)], \dots, E[\tilde{\theta}_a^r(t)]\}$, where $E[\tilde{\theta}_a^k(t)]$ denotes the expected value of random variable $\tilde{\theta}_a^k(t)$. For given node $i \in \mathcal{V}$ and departure time $t \in \mathcal{S}$, hyperpath a is Pareto-optimal if no other path $b \in H(t)$ exists

such that

$$E[\tilde{\theta}_b^k(t)] \leq E[\tilde{\theta}_a^k(t)] \quad \forall k \in \{1,2,\dots,r\} \text{ and } \exists h \in \{1,2,\dots,r\} \text{ such that}$$

$$E[\tilde{\theta}_b^h(t)] < E[\tilde{\theta}_a^h(t)].$$

The second problem seeks a single best hyperpath with respect to a linear disutility function from each node to a specified destination for each departure time $t \in S$. Hyperpath $f \in H(t)$ is the least expected disutility (LED) hyperpath for departure time t if

$$E[U_f(t)] = \min_{g \in H(t)} E[U_g(t)],$$

where $E[U_g(t)] = \sum_{k=1}^r w^k \cdot E[\tilde{\theta}_g^k(t)]$ and w^k is the weight assigned to criterion k .

2.5 SOLUTION APPROACHES

In this section, exact algorithms are presented for generating all adaptive Pareto-optimal paths in MSTV networks. In Subsection 2.5.1, the Adaptive Pareto-Optimal Strategy (APS) algorithm is proposed to generate all Pareto-optimal hyperpaths for every departure time from every node to a select destination. In Subsections 2.5.2 and 2.5.3, the Adaptive Least Expected Disutility Strategy I & II (ALEDS I & II) algorithms are developed. These algorithms find a single “best compromise” hyperpath that minimizes the expected disutility given a linear utility function. The proposed algorithms can be viewed as extensions of the Expected Lower Bound (ELB) algorithm for finding adaptive LET paths in single criterion STV networks (Miller-Hooks and Mahmassani, 2000). The ELB algorithm and these extensions are specialized label-correcting algorithms. Description of each algorithm, the specific

procedural steps and associated proofs are provided.

2.5.1 The Adaptive Pareto-Optimal Strategy (APS) Algorithm

The APS algorithm extends the Expected Lower Bound (ELB) algorithm of Miller-Hooks and Mahmassani (2000) for finding adaptive LET paths in single criterion STV networks for use with multiple criteria. In the ELB algorithm, prior to termination, a label is associated with each node and each departure time, each of which represents an upper bound on the expected travel time from that node to the destination for that departure time. Upon termination, each label provides the LET to the desired destination. Unlike solutions to the single criterion adaptive path problem, where a single hyperpath exists at each node and departure time, in the multicriteria adaptive path problem, multiple Pareto-optimal hyperpaths may exist at each node and departure time. Moreover, for each node and departure time, rather than computing a single expected value, r expected values must be maintained (i.e. one for each criterion). Similar to the single criterion problem, where a particular hyperpath may only be optimal at a particular departure time, in the multiobjective problem, a hyperpath that is Pareto-optimal at one departure time may be dominated at another departure time. The computation of these hyperpaths requires path information only at departure times at which these hyperpaths are non-dominated. Thus, one need only maintain the hyperpaths at the departure times at which they are non-dominated.

For each node $i \in \mathcal{V}$, each departure time $t \in \mathcal{S}$ and each currently Pareto-optimal hyperpath x to the destination, a vector label $\lambda_{ix}(t) = \{\lambda_{ix}^k(t)\}_{k \in R}$ is maintained, where each element of vector label component $\lambda_{ix}^k(t)$ is the expected

value with respect to criterion k along potentially Pareto-optimal hyperpath x from node i at departure time t to the destination node N . Until the algorithm terminates, multiple vector labels are maintained at each node and departure time. In an intermediate iteration of the APS algorithm, $X_i(t)$ contains labels of currently Pareto-optimal hyperpaths for node $i \in \mathcal{V}$ at departure time $t \in \mathcal{S}$. $|X_i(t)|$ is equal to the number of currently Pareto-optimal vector labels maintained for node i at time t . It is assumed that $\forall t \in \mathcal{S}, i \in \mathcal{V}, k \in R, 0 < \varepsilon < \Delta t, \lambda_{ix}^k(t + \varepsilon) = \lambda_{ix}^k(t)$. For each departure time occurring after the peak period, $t > t_0 + I\Delta t, \lambda_{ix}^k(t) = \lambda_{ix}^k(t_0 + I\Delta t)$. Figure 2.3 illustrates the structure of the vector labels at each departure time and demonstrates that for each departure time, more than one vector label, each associated with different hyperpaths, may be maintained. Assume the period of interest is discretized into four intervals, $\Delta t = 1$, and there are two Pareto-optimal hyperpaths at departure times 0, 1 and 3 and one at time 2. That is, $X_i(0) = X_i(1) = X_i(3) = \{1, 2\}$ and $X_i(2) = \{1\}$.

A temporary vector label, $\eta_i(t) = \{\eta_i^k(t)\}_{k \in R}$, is employed. To evaluate whether or not a newly constructed hyperpath is dominated, the temporary vector label is compared with the labels of the currently Pareto-optimal hyperpaths at node i and time t . If the temporary hyperpath is dominated by one or more of the currently Pareto-optimal hyperpaths, it is discarded. Likewise, if it dominates one or more of the currently Pareto-optimal hyperpaths, the temporary label is maintained and the labels associated with the dominated hyperpaths are discarded.

Time	Position 1	Position 2
0	$\lambda_{i1}(0) = \{ \lambda_{i1}^1(0), \lambda_{i1}^2(0), \dots, \lambda_{i1}^r(0) \}$	$\lambda_{i2}(0) = \{ \lambda_{i2}^1(0), \lambda_{i2}^2(0), \dots, \lambda_{i2}^r(0) \}$
1	$\lambda_{i1}(1) = \{ \lambda_{i1}^1(1), \lambda_{i1}^2(1), \dots, \lambda_{i1}^r(1) \}$	$\lambda_{i2}(1) = \{ \lambda_{i2}^1(1), \lambda_{i2}^2(1), \dots, \lambda_{i2}^r(1) \}$
2	$\lambda_{i1}(2) = \{ \lambda_{i1}^1(2), \lambda_{i1}^2(2), \dots, \lambda_{i1}^r(2) \}$	–
3	$\lambda_{i1}(3) = \{ \lambda_{i1}^1(3), \lambda_{i1}^2(3), \dots, \lambda_{i1}^r(3) \}$	$\lambda_{i2}(3) = \{ \lambda_{i2}^1(3), \lambda_{i2}^2(3), \dots, \lambda_{i2}^r(3) \}$

Figure 2.3. Illustration of Vector Labels at Node i.

Similar to the ELB algorithm, the APS algorithm proceeds in an iterative manner by scanning a node from a scan eligible (SE) list, working backward starting from the destination node. The ELB algorithm builds a hyperpath from each node at each departure time through the currently LET subhyperpaths associated with possible arrival times at a successor node. Only one hyperpath, i.e. the one with the LET, will be maintained. The APS algorithm, on the other hand, may build more than one hyperpath from each node at each departure time. That is, hyperpaths are constructed through each of the currently Pareto-optimal subhyperpaths at a successor node. These hyperpaths are examined to determine whether or not they are dominated and only the non-dominated hyperpaths will be maintained.

To construct a vector label associated with a single hyperpath from node i at departure time t through successor node j , employing arc (i,j) , the vector label of one subhyperpath $\lambda_{jx}(t + c_{ij}^{1z_1}(t))$, $x \in X_j(t + c_{ij}^{1z_1}(t))$ for each value of $z_1 \in \{1,2,\dots,D\}$, i.e. for a given travel time on arc (i,j) , $c_{ij}^{1z_1}(t)$, must be selected. Each combination of z_1 and $\lambda_{jx}(t + c_{ij}^{1z_1}(t))$ is referred to by the pair (z_1, x) . The hyperpath is constructed

from D such pairs (one for each possible travel time on arc (i,j)). Because there may be more than one combination of (z_1, x) pairs when $|X_j(t + c_{ij}^{1z_1}(t))| > 1$ for at least one possible arrival time at node j , it may be possible to construct more than one hyperpath from node i . In fact, there are exactly $\prod_{z=1}^D |X_j(t + c_{ij}^{1z_1}(t))|$ hyperpaths that can be constructed. An example is given to illustrate the hyperpath construction in Figure 2.4.

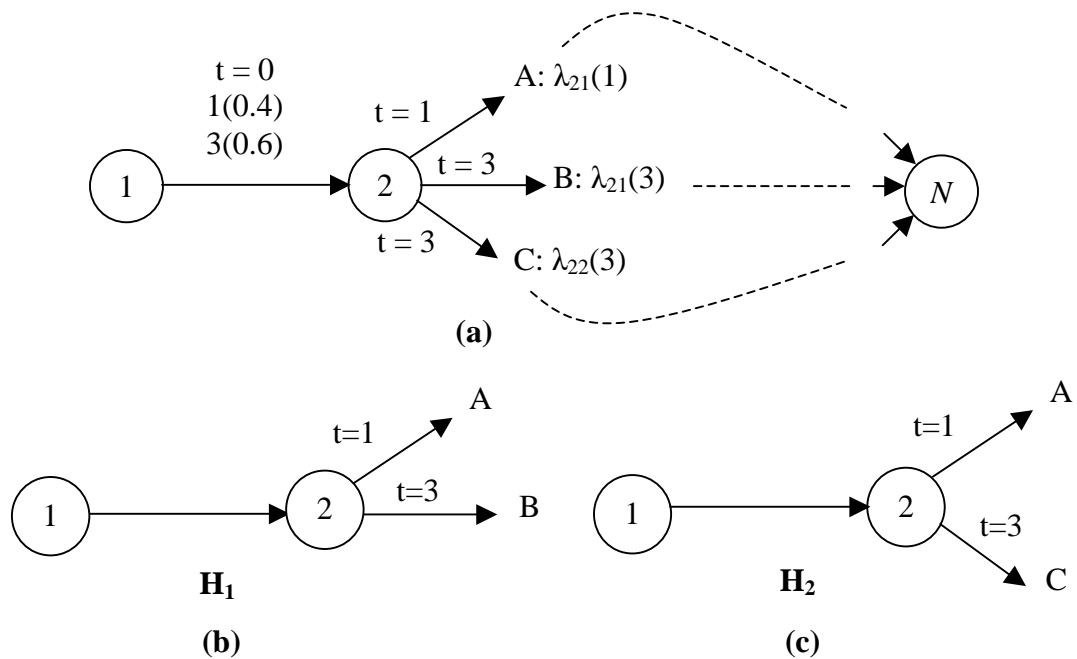


Figure 2.4. Example for the APS algorithm.

Figure 2.4 depicts a set of possible hyperpaths linking the origin, node 1, to the destination, node N in a MSTV network. There are two possible travel times on arc $(1,2)$ at departure time $t = 0$: 1 with probability 0.4 and 3 with probability 0.6, leading to two corresponding arrival times at node 2: $t = 1$ and $t = 3$. Suppose at node

2, there exist hyperpath A for time $t = 1$ with associated vector label $\lambda_{21}(1)$ and two Pareto-optimal hyperpaths B and C for time $t = 3$ with associated vector labels $\lambda_{21}(3)$ and $\lambda_{22}(3)$, respectively. That is, $X_2(1) = \{1\}$ and $X_2(3) = \{1,2\}$. Then, at node 1, two different combinations of (z_1, x) pairs exist, resulting in the generation of two hyperpaths at node 1 for departure time $t = 0$:

$$H_1: (1, 1), (2, 1)$$

$$H_2: (1, 1), (2, 2)$$

For H_2 in this example, two combinations contribute to the computation of the expected travel time: (1) the first possible travel time on arc (1,2) and the first vector label at node 2 at the arrival time corresponding to this first travel time; and (2) the second possible travel time on arc (1,2) and the second vector label at node 2 at the arrival time corresponding to this second travel time. These hyperpaths are depicted in Figure 2.4b and 2.4c. Let Q be the set of these (z_1, x) pairs comprising a single hyperpath H , i.e. for H_2 , $Q = \{(1, 1), (2, 2)\}$. The expected travel time (i.e. criterion 1) for H_2 can be computed as follows.

$$\begin{aligned} \eta_1^1(0) &= \sum_{(z_1, x) \in Q} [c_{12}^{1z_1}(0) + \lambda_{2x}^1(0 + c_{12}^{1z_1}(0))] \cdot \rho_{12}^{1z_1}(0) \\ &= [c_{12}^{11}(0) + \lambda_{21}^1(0 + c_{12}^{11}(0))] \cdot \rho_{12}^{11}(0) + [c_{12}^{12}(0) + \lambda_{22}^1(0 + c_{12}^{12}(0))] \cdot \rho_{12}^{12}(0). \end{aligned}$$

Note that two subhyperpaths, the first at arrival time $t = 1$ and the second at the arrival time $t = 2$ were employed in this computation.

To enable efficient reconstruction of the resulting hyperpaths after termination of the algorithm, two path pointers are employed for each node i at each time t along

hyperpath x . These pointers specify the successor node and specific subhyperpath at the successor node for each possible departure time. Specifically, for each vector label $x \in X_i(t)$, $\pi_{ix}(t)$ specifies the successor node to be taken from node i at departure time t along the hyperpath x . Unlike in the ELB algorithm where only one pointer is required for this purpose, in solving this multicriteria problem, a set $\{q_{ix}(t) = \{(z_1, x)\} \mid z_1 \in \{1, 2, \dots, D\}\}$ must be maintained to identify the appropriate subhyperpath at the successor node $\pi_{ix}(t)$ for each value of z_1 , i.e. for a particular arrival time at the successor node. Before a temporary vector label is checked for dominance, $\pi_{i0}(t)$ and $q_{i0}(t)$ are used to temporarily maintain the path information of that vector label. For the example in Figure 2.4, $\pi_{10}(0) = 2$ and $q_{10}(0) = \{(1, 1), (2, 2)\}$. The notation used in this section is summarized in Table 2.3.

Table 2.3. Notation employed in the APS algorithm.

	Notation
$X_i(t)$:	set of Pareto-optimal labels maintained at node i and departure time t
$\lambda_{ix}(t) = \{\lambda_{ix}^k(t)\}_{k \in R}$:	x^{th} vector label maintained in $X_i(t)$
$\lambda_{ix}^k(t)$:	expected value with respect to criterion k of a Pareto-optimal hyperpath from node i at departure time t to the destination node N
$\pi_{ix}(t)$:	successor node associated with $\lambda_{ix}(t)$ to be taken from node i at departure time t
$q_{ix}(t)$:	specific combination of subhyperpaths at node $\pi_{ix}(t)$ that is used to construct the hyperpath associated with $\lambda_{ix}(t)$
Q :	set of possible combinations of (z_1, x) pairs, where $z_1 = 1, 2, \dots, D$

The steps of the APS algorithm are provided next.

Algorithm APS

Step 1 (Initialization):

Initialize the elements of the vector labels and path pointers.

$$\lambda_{i1}^k(t) = \infty, \forall i \in \mathcal{V} \setminus N, k \in R, t \in \mathcal{S}.$$

$$\pi_{i1}(t) = \infty, \forall i \in \mathcal{V}, t \in \mathcal{S}.$$

$$q_{i1}(t) = \phi, \forall i \in \mathcal{V}, t \in \mathcal{S}.$$

$$X_i(t) = \{1\}, \forall i \in \mathcal{V}, t \in \mathcal{S}.$$

$$\lambda_{N1}^k(t) = 0, \forall k \in R, t \in \mathcal{S}.$$

Create the SE list and insert the destination node N into the SE list.

Step 2 (Select Node for Scanning):

If the SE list is empty, go to step 4. Otherwise, select and delete a node from the SE list. Call this node the current node j .

Step 3 (Update the Node Labels):

For each $i \in \Gamma^{-1}(j)$,

For each $t \in \mathcal{S}$,

Identify $\{c_{ij}^{1z_1}(t)\}_{z_1=\{1,2,\dots,D\}}$ and all possible Q combinations of $\{(z_1, x)\}_{z_1=\{1,2,\dots,D\}}$.

For each combination Q , compute temporary label value $\eta_i(t)$ as follows:

for travel time ($k = 1$):

$$\eta_i^1(t) = \sum_{(z_1, x) \in Q} [c_{ij}^{1z_1}(t) + \lambda_{jx}^1(t + c_{ij}^{1z_1}(t))] \cdot \rho_{ij}^{1z_1}(t)$$

for other criteria ($k \neq 1$):

$$\eta_i^k(t) = \sum_{(z_1, x) \in Q} \sum_{z_k=1}^D [c_{ij}^{kz_k}(t) + \lambda_{jx}^k(t + c_{ij}^{1z_1}(t))] \cdot \rho_{ij}^{1z_1}(t) \cdot \rho_{ij}^{kz_k}(t).$$

Set $\pi_{i0}(t) = j$ and $q_{i0}(t) = Q$.

Dominance Check

Check if this newly generated hyperpath with associated label $\eta_i(t)$ is dominated through pairwise comparison to all other non-dominated labels in $X_i(t)$ (see condition (1)):

If the label $\eta_i(t)$ is dominated, discard it.

Otherwise, add this label to $X_i(t)$ and remove all labels of dominated hyperpaths from $X_i(t)$.

$SE = SE \cup \{i\}$.

Return to step 2.

Step 4 (Termination):

Stop.

The algorithm terminates with all Pareto-optimal hyperpaths with respect to the expected value of each criterion from each origin to the destination node N , for each departure time $t \in S$. The procedural steps of this algorithm are illustrated on an example problem in Appendix A.

In STV networks for a single criterion *a priori* path problem, all subpaths of a non-dominated path (with respect to a variety of different dominance criteria) with the same destination node as this path must themselves be non-dominated (Miller-Hooks, 1997). This concept is extended here to multiple criteria in STV networks with the following lemma.

Lemma 2.1. For a given departure time $t \in \mathcal{S}$, any hyperpath that contains a dominated subhyperpath to the same destination as this hyperpath is itself dominated.

Proof (by counter example). Suppose vector label $\lambda_{iu}(t)$ associated with a hyperpath at node $i \in \mathcal{V}$ for departure time $t \in \mathcal{S}$ is non-dominated, i.e. $u \in X_i(t)$. Further, suppose that this hyperpath contains a subhyperpath from node $j \in \Gamma^{+1}(i)$ employed at departure time $s > t$ with associated vector label $\lambda_{jy}(s)$, $y \notin X_j(s)$ that is dominated by another non-dominated subhyperpath at the same departure time, $\lambda_{jw}(s)$, $w \in X_j(s)$. Without loss of generality, assume the following condition holds:

$$\lambda_{jw}^k(s) = \lambda_{jy}^k(s) \forall k \in \{1, 2, \dots, h-1, h+1, \dots, r\} \text{ and } \lambda_{jw}^h(s) < \lambda_{jy}^h(s) \text{ (condition 2).}$$

We proceed by showing that if the hyperpath at node i and departure time t with associated vector label $\lambda_{iu}(t)$ contains this dominated subhyperpath with associated vector label $\lambda_{jy}(s)$, it must be dominated, contrary to our assumption that it is non-dominated. Rather, another hyperpath will exist from node i at departure time t constructed through the vector label $\lambda_{jw}(s)$ that dominates it.

Assume $\lambda_{iu}(t)$ is computed from a certain Q combination of (z_1, x) pairs, where $z_1 = \{1, 2, \dots, g-1, g, g+1, \dots, D\}$, $x \in X_j(t + c_{ij}^{1z_1}(t))$. Included in this combination is pair (g, y) , where g is chosen such that $t + c_{ij}^{1g}(t) = s$. Since there exists a subhyperpath at node j and time s with vector label $\lambda_{jw}(s)$ that dominates $\lambda_{jy}(s)$, another hyperpath from node i can be constructed that employs $\lambda_{jw}(s)$ in place of $\lambda_{jy}(s)$, i.e. we replace (g, y) in Q with (g, w) , all else being equal. Let the vector label corresponding to this hyperpath be $\lambda_{iv}(t)$, $v \neq u$. By condition (2), we can conclude that $\lambda_{iv}(t)$ will dominate $\lambda_{iu}(t)$. This contradicts our assumption that the hyperpath associated with $\lambda_{iu}(t)$ at node i for departure time t is non-dominated. A hyperpath with a dominated subhyperpath will itself be dominated. \blacklozenge

Proposition 2.6. Upon termination, the APS algorithm generates all Pareto-optimal hyperpaths.

Proof. We begin by showing that any final label generated by the algorithm is, in fact, Pareto-optimal. Let $\lambda_{ix}(t)$, be a label associated with one of the Pareto-optimal hyperpaths for departure time t from node i determined by the algorithm (i.e. $x \in X_i(t)$). No other hyperpath with associated label $v \notin X_i(t)$ can exist such that

$$\lambda_{iv}^k(t) \leq \lambda_{ix}^k(t) \text{ for all } k \in \{1, 2, \dots, r\} \text{ and } \lambda_{iv}^h(t) < \lambda_{ix}^h(t) \text{ for some } h \in \{1, 2, \dots, r\}$$

(condition 3).

Suppose there exists such a hyperpath with vector label $\lambda_{iv}(t)$ for which condition (3) holds. Then, one of the following must be true: 1) $\lambda_{iv}(t)$ was dominated by another label or 2) $\lambda_{iv}(t)$ was never constructed in step 3 of the algorithm. If there exists a hyperpath that dominates the hyperpath associated with $\lambda_{iv}(t)$, it would also dominate the hyperpath associated with $\lambda_{ix}(t)$, contradicting the assumption that the hyperpath associated with $\lambda_{ix}(t)$ is Pareto-optimal. Thus, $\lambda_{iv}(t)$ must not have been constructed. If $\lambda_{iv}(t)$ was never constructed, either it contains a subhyperpath that is dominated and thus, by Lemma 2.1, is dominated or the SE list cannot be empty, contradicting the assumption of termination. This establishes that all solution hyperpaths in the final solution set are Pareto-optimal. One must next establish that all Pareto-optimal hyperpaths are generated.

Assume there exists a hyperpath for some departure time that is not dominated by any other hyperpath, but that is not present in the final set of the Pareto-optimal set of solutions. This hyperpath could only be left out of the solution set if it was never constructed. That is, a subhyperpath of this hyperpath must be dominated or a subhyperpath of this path was never constructed. In the former case, if the subhyperpath is dominated, then by Lemma 2.1 any hyperpath containing this subhyperpath must be dominated, a contradiction. The latter case could occur only if (1) the subhyperpath contains its own subhyperpath that has been dominated, and thus, it would be dominated, or (2) there is no path between the origin of the subhyperpath and the destination node, and thus, the hypothetical hyperpath could not exist. Hence, no path outside the final solution set can be Pareto-optimal. \blacklozenge

Proposition 2.7. The APS algorithm has exponential worst-case computational complexity.

Discussion. As discussed in Section 2.1, it is possible that in the worst-case all possible hyperpaths are Pareto-optimal. Consequently, the APS algorithm, which seeks all Pareto-optimal hyperpaths, is exponential in worst-case computational complexity.

2.5.2 The Adaptive Least Expected Disutility Strategy I (ALEDS I) Algorithm

The APS algorithm described in Subsection 2.5.1 generates all Pareto-optimal hyperpaths. While a decision-maker could *a posteriori* select a single “best” solution from among all Pareto-optimal hyperpaths, the generation of all such hyperpaths may require enormous computational effort. Thus, in this subsection, an algorithm is presented to efficiently generate a single “best” hyperpath by explicitly representing the decision maker’s preference structure through a disutility function. Instead of constructing multiple vector labels for all Pareto-optimal hyperpaths, this algorithm maintains only one vector label $\lambda_i(t) = \{\lambda_i^k(t)\}_{k \in R}$, at each node $i \in \mathcal{V}$ and each departure time $t \in \mathcal{S}$, where $\lambda_i^k(t)$ indicates the expected value for the path attribute with respect to criterion k . For each $i \in \mathcal{V}$ and $t \in \mathcal{S}$, a label $U_i(t)$ is employed. The disutility function $U_i(t)$ is assumed to be linear and can be written as follows:

$$U_i(t) = \sum_{k=1}^r w^k \cdot \lambda_i^k(t) \quad (\text{equation 1}),$$

where w^k is the weight for criterion k based on the traveler’s preferences and

$\sum_{k=1}^r w^k = 1$. Prior to termination, $U_i(t)$ provides an upper bound on the expected disutility for traveling from node i at departure time t to the destination. Upon termination of the algorithm, it maintains the least expected disutility (LED) of any hyperpath from node i at departure time t .

Like the APS algorithm, this algorithm is an extension of the ELB algorithm. It works iteratively by selecting a node from a SE list, working backward from the destination. At each node, the expected values for every criterion are computed and the disutility is calculated using equation (1). Temporary labels $\eta_i^k(t)$ and $\nu_i(t)$ maintain these values prior to updating $\lambda_i^k(t)$ and $U_i(t)$, respectively. If $\nu_i(t) < U_i(t)$, then $U_i(t)$ is set to $\nu_i(t)$ and $\lambda_i^k(t) \forall k \in R$ are updated accordingly. Upon termination, $\{\lambda_i^k(t)\}_{k \in R}$ contains the expected values for every criterion of the hyperpath with associated LED $U_i(t)$ from node i at departure time t to the destination N . To construct the LED hyperpaths efficiently, a single pointer, $\pi_i(t)$, is required to specify the successor node for travel from node i at departure time t .

Table 2.4. Notation employed in the ALEDS I and II algorithms.

	Notation
$\lambda_i(t) = \{\lambda_i^k(t)\}_{k \in R}$	vector label associated with node i at time t
$\lambda_i^k(t)$	expected value with respect to criterion k of the LED hyperpath from node i at departure time t to the destination node N
$\pi_i(t)$	successor node to be taken from node i at departure time t
w^k	weight for criterion k
$U_i(t)$	LED for traveling from node i at departure time t to the destination N

The procedural steps of the ALEDS I are given hereafter.

Algorithm ALEDS I

Step 1 (Initialization):

Initialize the elements of the vector labels and path pointers.

$$\lambda_i^k(t) = \infty, \forall i \in \mathcal{V} \setminus N, k \in R, t \in \mathcal{S}.$$

$$U_i(t) = \infty, \forall i \in \mathcal{V} \setminus N, t \in \mathcal{S}.$$

$$\pi_i(t) = \infty, \forall i \in \mathcal{V}, t \in \mathcal{S}.$$

$$\lambda_N^k(t) = 0, \forall k \in R, t \in \mathcal{S}.$$

$$U_N(t) = 0, \forall t \in \mathcal{S}.$$

Create the SE list and insert the destination node N into the SE list.

Step 2 (Select Node for Scanning):

If the SE list is empty, go to step 4. Otherwise, select and delete a node from the SE list. Call this node the current node j .

Step 3 (Update the Node Labels):

For each $i \in \Gamma^{-1}(j)$,

For each $t \in \mathcal{S}$,

$$v_i(t) = \sum_{k=1}^r w^k \cdot \eta_i^k(t),$$

where

$$k = 1: \eta_i^1(t) = \sum_{z_1=1}^D [c_{ij}^{1z_1}(t) + \lambda_j^1(t + c_{ij}^{1z_1}(t))] \cdot \rho_{ij}^{1z_1}(t)$$

$$k \neq 1: \eta_i^k(t) = \sum_{z_1=1}^D \sum_{z_k=1}^D [c_{ij}^{kz_k}(t) + \lambda_j^k(t + c_{ij}^{1z_1}(t))] \cdot \rho_{ij}^{1z_1}(t) \cdot \rho_{ij}^{kz_k}(t).$$

If $v_i(t) < U_i(t)$, then $U_i(t) = v_i(t)$, $\pi_i(t) = j$, $\lambda_i^k(t) = \eta_i^k(t)$ for $\forall k \in R$,

and $SE = SE \cup \{i\}$.

Return to step 2.

Step 4 (Termination):

Stop.

The algorithm results in the LED hyperpaths from each origin to the destination node N , for each departure time $t \in \mathcal{S}$, for a given set of criterion weights w^k and a linear utility function.

Proposition 2.8. The worst-case computational complexity of the ALEDS I algorithm with FIFO (first-in, first-out) SE list is $\sim O(V^3 \cdot I^2 \cdot P^2 \cdot R)$, where $V = |\mathcal{V}|$ is the number of nodes in the network, I is the number of time intervals within the period of interest, $P = |\mathcal{P}|$ is the maximum number of possible arc weights for a given criterion and $R = |\mathcal{R}|$ is the number of criteria.

Discussion. The worst-case computational complexity of this algorithm can be derived in a similar manner as was the ELB algorithm in Miller-Hooks and Mahmassani (2000). The only difference is that step 3 of the ELB algorithm has worst-case computational complexity $\sim O(V \cdot I \cdot P)$, whereas step 3 of the APS algorithm has complexity $\sim O(V \cdot I \cdot P^2 \cdot R)$. Therefore, the proposed algorithm has worst-case computational complexity $\sim O(V^3 \cdot I^2 \cdot P^2 \cdot R)$, $O(P \cdot R)$ worse than that of the ELB

algorithm.

Proof of correctness will be given in Section 2.5.3.

2.5.3 The Adaptive Least Expected Disutility Strategy II (ALEDS II) Algorithm

In this subsection, an improvement to the ALEDS I algorithm is presented, referred to as the ALEDS II algorithm. In the ALEDS I algorithm, the expected value for each criterion was computed before computing the disutility for the associated hyperpath. To correctly accomplish this, all associated path labels that result in the LED needed to be maintained. The ALEDS II algorithm, however, assesses the expected disutility directly and keeps only the minimum value for each node and departure time. Thus, there is no need to maintain the individual labels $\lambda_i(t) = \{ \lambda_i^k(t) \}_{k \in R}, \forall i \in \mathcal{V}, t \in \mathcal{S}$, which are needed in the first variation of the algorithm. While the ALEDS I algorithm is more intuitive, significant savings in computational complexity and storage requirements are achieved through the modifications employed in this second variation.

Lemma 2.2. The computation of $\nu_i(t)$ in step 3 of the ALEDS I algorithm can be completed via equation (2).

$$\nu_i(t) = \sum_{k=1}^r w^k \cdot \left[\sum_{z_k=1}^D c_{ij}^{kz_k}(t) \cdot \rho_{ij}^{kz_k}(t) \right] + \sum_{z_1=1}^D U_j(t + c_{ij}^{1z_1}(t)) \cdot \rho_{ij}^{1z_1}(t) \text{ (equation 2).}$$

Proof.
$$\nu_i(t) = \sum_{k=1}^r w^k \cdot \eta_i^k(t),$$

where

$$\eta_i^1(t) = \sum_{z_1=1}^D [c_{ij}^{1z_1}(t) + \lambda_j^1(t + c_{ij}^{1z_1}(t))] \cdot \rho_{ij}^{1z_1}(t)$$

$$\eta_i^k(t) = \sum_{z_1=1}^D \sum_{z_k=1}^D [c_{ij}^{kz_k}(t) + \lambda_j^k(t + c_{ij}^{1z_1}(t))] \cdot \rho_{ij}^{1z_1}(t) \cdot \rho_{ij}^{kz_k}(t), \text{ for } \forall k \in R \setminus 1.$$

Therefore,

$$\begin{aligned} v_i(t) &= w^1 \cdot \sum_{z_1=1}^D [c_{ij}^{1z_1}(t) + \lambda_j^1(t + c_{ij}^{1z_1}(t))] \cdot \rho_{ij}^{1z_1}(t) + \\ &\quad \sum_{k=2}^r w^k \cdot [\sum_{z_1=1}^D \sum_{z_k=1}^D [c_{ij}^{kz_k}(t) + \lambda_j^k(t + c_{ij}^{1z_1}(t))] \cdot \rho_{ij}^{1z_1}(t) \cdot \rho_{ij}^{kz_k}(t)] \\ &= w^1 \cdot \sum_{z_1=1}^D c_{ij}^{1z_1}(t) \cdot \rho_{ij}^{1z_1}(t) + \sum_{k=2}^r w^k \cdot [\sum_{z_1=1}^D \sum_{z_k=1}^D c_{ij}^{kz_k}(t) \cdot \rho_{ij}^{1z_1}(t) \cdot \rho_{ij}^{kz_k}(t)] + \\ &\quad w^1 \cdot \sum_{z_1=1}^D \lambda_j^1(t + c_{ij}^{1z_1}(t)) \cdot \rho_{ij}^{1z_1}(t) + \\ &\quad \sum_{k=2}^r w^k \cdot [\sum_{z_1=1}^D \sum_{z_k=1}^D \lambda_j^k(t + c_{ij}^{1z_1}(t)) \cdot \rho_{ij}^{1z_1}(t) \cdot \rho_{ij}^{kz_k}(t)] \\ &= w^1 \cdot \sum_{z_1=1}^D c_{ij}^{1z_1}(t) \cdot \rho_{ij}^{1z_1}(t) + \sum_{k=2}^r w^k \cdot [\sum_{z_k=1}^D c_{ij}^{kz_k}(t) \cdot \rho_{ij}^{kz_k}(t)] + \\ &\quad w^1 \cdot \sum_{z_1=1}^D \lambda_j^1(t + c_{ij}^{1z_1}(t)) \cdot \rho_{ij}^{1z_1}(t) + \sum_{k=2}^r w^k \cdot [\sum_{z_1=1}^D \lambda_j^k(t + c_{ij}^{1z_1}(t)) \cdot \rho_{ij}^{1z_1}(t)] \\ &= \sum_{k=1}^r w^k \cdot [\sum_{z_k=1}^D c_{ij}^{kz_k}(t) \cdot \rho_{ij}^{kz_k}(t)] + \sum_{k=1}^r w^k \cdot [\sum_{z_1=1}^D \lambda_j^k(t + c_{ij}^{1z_1}(t)) \cdot \rho_{ij}^{1z_1}(t)] \\ &= \sum_{k=1}^r w^k \cdot [\sum_{z_k=1}^D c_{ij}^{kz_k}(t) \cdot \rho_{ij}^{kz_k}(t)] + \sum_{z_1=1}^D U_j(t + c_{ij}^{1z_1}(t)) \cdot \rho_{ij}^{1z_1}(t) \cdot \blacklozenge \end{aligned}$$

Lemma 2.2 shows that the ALEDS I algorithm can be simplified through more efficient computation of the expected disutility values. Identical solutions will be

produced by both ALEDS I and II algorithms. The algorithmic steps of this second variation are given next.

Algorithm ALEDS II

Step 1 (Initialization):

Initialize the labels and path pointers.

$$U_i(t) = \infty, \forall i \in \mathcal{V} \setminus N, t \in \mathcal{S}.$$

$$\pi_i(t) = \infty, \forall i \in \mathcal{V}, t \in \mathcal{S}.$$

$$U_N(t) = 0, \forall t \in \mathcal{S}.$$

Create the SE list and insert the destination node N into the SE list.

Step 2 (Select Node for Scanning):

If the SE list is empty, go to step 4. Otherwise, select and delete a node from the SE list. Call this node the current node j .

Step 3 (Update the Node Labels):

For each $i \in \Gamma^{-1}(j)$,

For each $t \in \mathcal{S}$,

$$v_i(t) = \sum_{k=1}^r w^k \cdot \left[\sum_{z_k=1}^D c_{ij}^{kz_k}(t) \cdot \rho_{ij}^{kz_k}(t) \right] + \sum_{z_1=1}^D U_j(t + c_{ij}^{1z_1}(t)) \cdot \rho_{ij}^{1z_1}(t).$$

If $v_i(t) < U_i(t)$, then $U_i(t) = v_i(t)$, and $\pi_i(t) = j$. $SE = SE \cup \{i\}$.

Return to step 2.

Step 4 (Termination):

Stop.

It is significant to note that both ALEDS I and II algorithms rely on the fact that a LED hyperpath contains only subhyperpaths with the LED to the same destination. The proof for this concept is given in Lemma 2.3.

Lemma 2.3. Any LED hyperpath contains only LED subhyperpaths to the same destination.

Proof (by counter example). Suppose the LED hyperpath associated with label $U_i(t)$ is constructed from some node i at some departure time $t \in S$ through node $j \in \Gamma^{+1}(i)$ (i.e. employing arc $(i,j) \in \mathcal{A}$). Assume there are D possible travel times on arc (i,j) for departure time $t \in S$ (i.e. $z_1 = \{1,2,\dots,D\}$), resulting in D possible arrival times at node j :

$$s_1 = t + c_{ij}^{11}(t), s_2 = t + c_{ij}^{12}(t), \dots, s_D = t + c_{ij}^{1D}(t).$$

And, suppose that the LED hyperpath from node i at departure time t contains a subhyperpath at node j with associated label $U_j(s_q)$ for $q \leq D$ that is not the LED hyperpath from node j at departure time s_q .

$$\begin{aligned} U_i(t) &= \sum_{k=1}^r w^k \cdot \left[\sum_{z_k=1}^D c_{ij}^{kz_k}(t) \cdot \rho_{ij}^{kz_k}(t) \right] + \sum_{z_1=1}^D U_j(t + c_{ij}^{1z_1}(t)) \cdot \rho_{ij}^{1z_1}(t) \\ &= \sum_{k=1}^r w^k \cdot \left[\sum_{z_k=1}^D c_{ij}^{kz_k}(t) \cdot \rho_{ij}^{kz_k}(t) \right] \\ &\quad + U_j(s_q) \cdot \rho_{ij}^{11}(t) + \sum_{z_1 \in \{1,2,\dots,q-1,q+1,\dots,D\}} U_j(t + c_{ij}^{1z_1}(t)) \cdot \rho_{ij}^{1z_1}(t). \end{aligned}$$

If the label component of the LED hyperpath from node j at time s_q is given by

$U_j^*(s_q)$, then $U_j^*(s_q) < U_j(s_q)$. If the LED hyperpath from node j is employed in place of this other hyperpath, and thus replacing $U_j(s_q)$ with $U_j^*(s_q)$, all else being equal, a lower value of expected disutility, $U_i(t)$, will be obtained. This contradicts our assumption that the LED hyperpath from node i at departure time t can contain a subhyperpath through node $j \in \Gamma^{+1}(i)$ that is not the LED hyperpath from node j at one of the possible arrival times. Since one can extend this logic to any node contained in the LED hyperpath from node i and for any node i at any departure time t , this establishes that any LED hyperpath contains only LED subhyperpaths to the same destination. ♦

Proposition 2.9. Upon termination, the ALEDS I and II algorithms provide the LED hyperpaths for each node and each departure time in the period of interest.

Proof. Assume a hyperpath associated with $U_i(t)$ exists in the final set of solutions such that

$$U_i(t) > \sum_{k=1}^r w^k \cdot \left[\sum_{z_k=1}^D c_{ij}^{kz_k}(t) \cdot \rho_{ij}^{kz_k}(t) \right] + \sum_{z_1=1}^D U_j(t + c_{ij}^{1z_1}(t)) \cdot \rho_{ij}^{1z_1}(t) \text{ for some } j \in \Gamma^{+1}(i).$$

For this to be true, node j could not have been scanned and, hence, must still be in the SE list. Since this would contradict the assumption of termination of either algorithm, such a hyperpath could not result. Therefore, the ALEDS I and II algorithms terminate with the LED hyperpaths for each node at each departure time. ♦

Proposition 2.10. The worst-case computational complexity of the ALEDS II algorithm with FIFO SE list is $\sim O(V^3 \cdot I^2 \cdot P \cdot R)$.

Discussion. The complexity of the ALEDS II algorithm is $\sim O(P)$ better than that of the ALEDS I algorithm. This is because step 3 has complexity $\sim O(V \cdot I \cdot P \cdot R)$ as compared with $\sim O(V \cdot I \cdot P^2 \cdot R)$ of the ALEDS I algorithm.

Note that the ALEDS II algorithm is only $\sim O(R)$ worse than the ELB algorithm of Miller-Hooks and Mahmassani (2000), which considers only one criterion.

2.6 NOTES ON ALGORITHM IMPLEMENTATION

In related problems that can be addressed by label correcting procedures (see for example Miller-Hooks, 1997), where multiple vector labels are associated with each node, it is sometimes advantageous to include node-label pairs in the SE list instead of only the nodes. While multiple vector labels are employed in the APS algorithm, because of the way that the labels are constructed, there is no benefit to including the node-label pairs in the SE list and, as is more commonly done, only the nodes are included in the list. A FIFO SE list structure is assumed in analyzing the complexity of the ALEDS algorithms; however, one could employ another structure, such as a deque implementation of the SE list. Additional information on structuring SE lists can be found in (Pape, 1974; Pallottino, 1984; Gallo and Pallottino, 1986; Ahuja et al., 1993; and Bertsekas et al., 1996). A reverse star representation of the networks is used (Ahuja et al., 1993).

2.7 COMPUTATIONAL EXPERIMENTS

In this section, results from computational experiments conducted on randomly generated networks with randomly generated time-varying PMFs of the arc attribute variables are given. The experiments were designed to evaluate the average computational performance of the proposed algorithms. The number of nodes (V), number of time intervals (I), number of elements in the PMFs (P) and number of criteria (R) are predesignated. In accordance with transportation networks, the in-degree and out-degree are both four, on average, and range between 2 and 9. The same methodology used for creating the STV networks as described in Miller-Hooks and Mahmassani (2000) is extended for use in MSTV networks and is applied here. That is, for each criterion, a uniform distribution was fixed with a lower bound of one unit. The upper bound was designed to linearly increase from 5 to 10 units in the first half of the peak period and then linearly decrease to 5 units in the second half of the peak period.

2.7.1 Experimental Design

The algorithms were coded in C++ and run on a DEC Alpha XP1000 professional workstation with 1 gigabyte ram and 2 gigabyte swap, running Digital 4.0E operating system, using Digital's C++ compiler. Three sets of experiments were conducted. First, the average performance in terms of run time of the APS algorithm was tested. Networks consisting of 25 nodes, peak periods of 15 and 30 time intervals, 3 elements in each PMF and 2 criteria were considered. Second, the performance of the ALEDS I and II algorithms in terms of average run time was compared through experiments on

eight networks with either 100 or 500 nodes, 60 or 120 time intervals, 5 elements in each PMF and either 2 or 6 criteria. Finally, additional tests of the ALEDS II algorithm were conducted on networks consisting of 50, 100, 500 and 1,000 nodes, a peak period of 15, 30, 60 and 120 time intervals, three levels of the number of elements in the PMFs (5, 10 and 20) and three levels of the number of criteria (2, 4 and 6). In all of these experiments, a FIFO SE list was employed. For each network and configuration of number of time intervals in the peak period, number of elements in the PMFs and number of criteria, 30 runs were completed, corresponding to 30 randomly selected destinations. The average of these 30 runs is reported.

2.7.2 Average Run Times of the APS Algorithm

Average run times over the 30 runs on each network configuration are given in Table 2.5.

Table 2.5. Average run times in c.p.u. seconds for the APS algorithm on a 25 node network.

Test	I	P	R	Average run time (c.p.u. seconds)
1	15	3	2	26.07
2	15	3	4	520.9
3	15	5	2	411.6
4	30	3	2	234.8

Because many Pareto-optimal hyperpaths exist for each network, the APS algorithm performed very poorly in these experiments in terms of run time and memory requirements. Thus, more extensive testing on larger networks was not completed. As suggested by Table 2.5, the average run times increase considerably

with the number of criteria and the number of elements in the PMFs, as predicted by the worst-case computational analysis.

2.7.3 Average Run Times of the ALEDS I and II Algorithms

In this subsection, the improvements in average computational time attained by the use of the ALEDS II algorithm over the ALEDS I algorithm are examined through tests on several network configurations. Average run times resulting from the experiments are shown in Table 2.6.

Table 2.6. Average run times in c.p.u. seconds for the ALEDS I and II algorithms.

P = 5				
V	I	R	ALEDS I	ALEDS II
100	60	2	0.074	0.035
		6	0.153	0.056
	120	2	0.157	0.072
		6	0.304	0.112
500	60	2	0.333	0.195
		6	0.784	0.292
	120	2	0.68	0.393
		6	1.557	0.574

From the table, the ALEDS II algorithm is approximately 1 to 2 times faster than the ALEDS I algorithm for the problems with 2 criteria and 2 to 3 times faster for the problems with 6 criteria. Note that for the 25 node network with 3 elements in the PMFs, 15 time intervals in the peak period, and two criteria, the average and maximum numbers of actual Pareto-optimal hyperpaths (as generated by the APS algorithm) over all nodes and departure times are 3.3 and 119, respectively. Thus, one can see that the ALEDS algorithms can provide significant savings in both

computational effort and memory requirements over procedures that generate all Pareto-optimal hyperpaths.

The computational performance of the ALEDS II algorithm was further tested through additional numerical experiments. The average run times over 30 destinations for the ALEDS II algorithm are summarized in Table 2.7. The results show that the ALEDS II algorithm performed well even on networks with 1000 nodes and better than the worst-case computational complexity, $\sim O(V^3 \cdot I^2 \cdot P \cdot R)$, given in Proposition 2.10. For instance, the average run time for the 1000 node network with 20 elements in the PMFs, 30 time intervals, 4 criteria is 0.629 c.p.u. seconds, which is less than twice the average run time of the same network with 2 criteria (requiring 0.453 c.p.u. seconds). Likewise, the average run time for the 500 node network with 20 elements in the PMFs, 30 time intervals, 4 criteria requires 0.298 c.p.u. seconds. Thus, the computational effort required by the 1000 node network was much less than the predicted worst-case of run time for the 500 node network. Despite this, as coded, experiments on some of the large networks could not be completed due to excessive memory requirements. Such requirements arise, at least in part, as a consequence of the use of four-dimensional arrays (node, time, criterion, label number) used to implement the vector labels. More efficient coding of the algorithm may enable solution of these larger size problems.

Table 2.7. Average run times in c.p.u. seconds for the ALEDS II algorithm.

R = 2					
V	P	I = 15	I = 30	I = 60	I = 120
50	5	0.003	0.007	0.016	0.034
	10	0.005	0.012	0.025	0.052
	20	0.009	0.018	0.040	0.082
100	5	0.007	0.017	0.037	0.071
	10	0.014	0.027	0.055	0.108
	20	0.019	0.041	0.083	0.164
500	5	0.056	0.110	0.217	0.432
	10	0.077	0.148	0.292	0.539
	20	0.109	0.212	0.413	0.865
1000	5	0.134	0.255	0.503	0.933
	10	0.173	0.331	0.629	-
	20	0.242	0.453	0.890	-
R = 4					
V	P	I = 15	I = 30	I = 60	I = 120
50	5	0.004	0.010	0.021	0.044
	10	0.007	0.017	0.035	0.070
	20	0.013	0.028	0.057	0.112
100	5	0.010	0.022	0.045	0.090
	10	0.017	0.035	0.072	0.143
	20	0.028	0.057	0.113	0.225
500	5	0.068	0.131	0.255	0.503
	10	0.100	0.195	0.383	0.726
	20	0.153	0.298	0.587	-
1000	5	0.154	0.297	0.572	1.071
	10	0.219	0.424	0.793	-
	20	0.324	0.629	-	-
R = 6					
V	P	I = 15	I = 30	I = 60	I = 120
50	5	0.006	0.013	0.027	0.056
	10	0.010	0.022	0.043	0.090
	20	0.017	0.036	0.072	0.143
100	5	0.013	0.028	0.055	0.112
	10	0.022	0.044	0.089	0.179
	20	0.036	0.071	0.143	0.285
500	5	0.081	0.157	0.305	0.573
	10	0.119	0.240	0.458	-
	20	0.190	0.372	0.744	-
1000	5	0.179	0.347	0.641	-
	10	0.265	0.503	-	-
	20	0.4	0.787	-	-

* - : memory requirements for reading the input files into variables were not sufficient

2.8 CONCLUSIONS

The problem of generating Pareto-optimal hyperpaths that seek to minimize the expected value of multiple criteria in MSTV networks is addressed in this chapter. An exact algorithm, the APS algorithm, is proposed for use in generating all Pareto-optimal solutions. Given multiple criteria, such solutions enable the driver to adaptively choose a path to travel at each intermediate location from among all non-dominated path strategies. As generation of all Pareto-optimal solutions may require generation of all solutions, the APS algorithm has exponential worst-case computational complexity. Hence, an efficient algorithm that can provide a single “best compromise” solution is required. Rather than generate all Pareto-optimal hyperpaths and *a posteriori* select a single solution, if a decision-maker’s preferences can be represented by a linear disutility function, a more direct and efficient approach is proposed (ALEDS I and II algorithms). While less intuitive than the ALEDS I algorithm, the ALEDS II algorithm provides substantial improvements in computational complexity and storage requirements.

The path strategies generated by the proposed algorithms enable travelers to dynamically choose their paths in response to knowledge of experienced traffic conditions. Consideration is given to the trade-offs among various attributes in the path selection process. Problems requiring the selection of such paths are encountered in a variety of application arenas, including selection of routes for emergency response units (medical, police, fire and other first responders), vehicles carrying hazardous materials, and individual travelers in congested city streets, as well as the selection of routes for data packets in data networks. In the context of supplying

routing instructions to drivers, it is assumed that the vehicles are equipped with on-board navigation systems. The proposed algorithms assume that estimates of future arc attribute values (i.e. their time-varying probability distributions) are known.

Computational experiments were conducted. Consistent with worst-case computational complexity, the results show that, in terms of average run times, the APS algorithm does not perform well in large networks. However, such an exact procedure can be quite useful in providing benchmark solutions on small problem instances when developing more efficient, but heuristic approaches. Results of computational experiments also show that the ALEDS II algorithm outperforms the ALEDS I algorithm. In addition, it appears that the average performance of the ALEDS II algorithm is better than predicted by worst-case computational complexity analyses.

CHAPTER 3

ADJUSTABLE PREFERENCE PATH STRATEGIES

3.1 INTRODUCTION

In this chapter, an efficient algorithm is proposed for determining adjustable preference path strategies (APPS) in MSTV transportation networks. The proposed algorithm determines adaptive path strategies that provide the next best move to take with respect to the expected value of a chosen criterion given the actual arrival time at an intermediate location. These strategies further permit a traveler to update his or her preference for a particular criterion to be used in selecting the remaining portion of the path while en route to the destination. Thus, a traveler can update his or her preference for which attribute of multiple attributes is of greatest importance and should be used in selecting the next best move. The traveler can then adaptively select the best path with respect to the selected criterion at each node in response to knowledge of experienced travel times on previously traveled arcs. The ability to change preferences in this way is referred to herein as adjustable preferences and adaptive strategies that allow for such adjustable preferences is referred to as adjustable preference path strategies (APPS).

The ability to update one's preference for a particular attribute while en route enables the traveler to respond to experienced travel conditions while traveling to the destination. Suppose at the outset the traveler has ample time to arrive at the desired location, and thus, chooses the path based on maximizing aesthetics, instead of minimizing travel time. Suppose the traveler discovers, en route, that the journey is

taking much longer than anticipated due to unexpected travel delays and minimizing travel time becomes more important. Allowing the traveler to adapt his/her path according to both revealed arrival times at intermediate locations and the traveler's changing preferences can lead to much more desirable path outcomes.

A MSTV example network shown in Figure 3.1 is given to illustrate the characteristics of the APPS in a MSTV network. Two criteria are considered: travel time and travel cost. Both attributes are assumed to be random quantities with probability distributions that vary with time. No waiting is allowed at any node and the arc attributes are assumed to be independent over space and time and independent of each other.

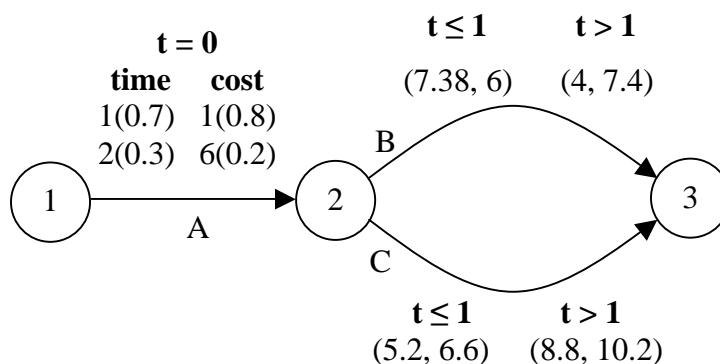


Figure 3.1. MSTV example network.

Two possible travel times on arc A when departing from node 1 at $t = 0$ are 1 with probability 0.7 and 2 with probability 0.3. There are also two possible costs: 1 with probability 0.8 and 6 with probability 0.2. For simplicity, the expected values are directly given for each attribute of subpaths B and C, e.g. the expected travel time and cost for path B at time $t \leq 1$ are 7.38 and 6, respectively.

The optimal APPS for traveling from node 1 at $t = 0$ to node 3 are portrayed in Figure 3.2. The traveler starts the trip at $t = 0$ along arc A with expected arrival time at node 2 of 6.14 units and expected cost of 8.42 units. The next move from node 2 depends on the actual arrival time and the traveler's preference for a particular criterion upon arrival at that node. If the arrival time is 1 and travel time is the preferred criterion at that time, subpath C is recommended. Subpath C incurs the least expected travel time (5.2 units). However, if expected cost is the preferred criterion, subpath B should be used with the least expected cost of 6 units. If the arrival time is 2, the APPS instruct the traveler to follow subpath B regardless of which criterion is preferred, because this subpath has both the least expected time and least expected cost.

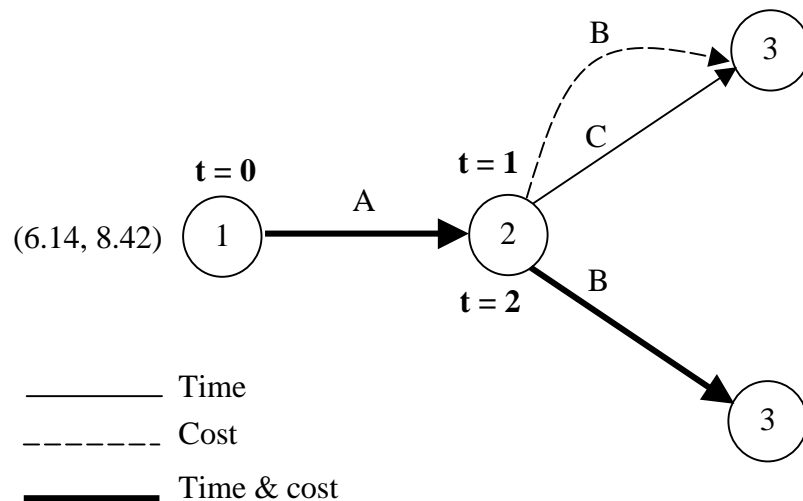


Figure 3.2. APPS solution.

In place of employing the proposed algorithm, identical solutions can be obtained by running, for example, the ELB algorithm (proposed by Miller-Hooks and Mahmassani (2000) for determining adaptive least expected time paths in STV networks) repeatedly, with a separate run for each criterion. The traveler could then choose which set of solutions to use whenever the traveler's preference for a particular attribute changes.

The primary contribution of this work is an algorithm that obviates the need for running the ELB (or similar) algorithm multiple times, leading to significant computational savings. Results of numerous numerical experiments show that the computational savings can be as large as 45% in comparison with the use of the ELB algorithm for problems with six criteria. In the next section, the problem definition and network notation employed in the proposed algorithm are provided.

3.2 NETWORK NOTATION AND PROBLEM DEFINITION

For consistency, the notation given in Chapter 2 is used herein. This chapter addresses the generation of the APPS for routing a traveler in MSTV networks. The adaptive path strategies that have the least expected value on a set of criteria are sought from each node to a specified destination for every departure time $t \in \mathcal{S}$. Let $H(t)$ be the set of all possible hyperpaths connecting an origin-destination pair for departure time t and let $\tilde{\theta}_a^k(t)$ be the random variable for the k^{th} criterion along a hyperpath $a \in H(t)$. $E[\tilde{\theta}_a(t)] = \{ E[\tilde{\theta}_a^1(t)], E[\tilde{\theta}_a^2(t)], \dots, E[\tilde{\theta}_a^k(t)], \dots, E[\tilde{\theta}_a^r(t)] \}$, where $E[\tilde{\theta}_a^k(t)]$ denotes the expected value of random variable $\tilde{\theta}_a^k(t)$. Hyperpath $b \in H(t)$ that satisfies the following condition is sought.

$$\exists h \in \{1, 2, \dots, r\} \text{ such that } E[\tilde{\theta}_b^h(t)] = \min_{\forall g \in H(t)} E[\tilde{\theta}_g^h(t)].$$

3.3 THE ADJUSTABLE PREFERENCE PATH STRATEGIES (APPS) ALGORITHM

The APPS algorithm extends the Expected Lower Bound (ELB) algorithm of Miller-Hooks and Mahmassani (2000) for finding adaptive least expected time paths in single criterion STV networks for use with multiple criteria. The algorithm determines the APPS in an iterative manner by scanning nodes from a scan eligible (SE) list working backward from the destination node. It proceeds by checking at each departure time whether using the hyperpaths associated with the scanned node (chosen in step 1 in the algorithm description) along the hyperpaths from its predecessor nodes generates a lower expected value of one or more of the criteria from these predecessor nodes to the destination than previously considered hyperpaths.

For each node $i \in \mathcal{V}$ at each departure time $t \in \mathcal{S}$, a vector label $\lambda_i(t) = \{\lambda_i^k(t)\}_{k \in \mathcal{R}}$, is maintained, where, prior to termination, $\lambda_i^k(t)$ provides an upper bound on the expected value with respect to criterion k for traveling from node i at departure time t to the destination node N . It is assumed that $\forall t \in \mathcal{S}, i \in \mathcal{V}, k \in \mathcal{R}, 0 < \varepsilon < \Delta t, \lambda_i^k(t + \varepsilon) = \lambda_i^k(t)$. For each departure time occurring after the peak period, $t > t_0 + I\Delta t, \lambda_i^k(t) = \lambda_i^k(t_0 + I\Delta t)$. Upon termination, for each $t \in \mathcal{S}$ and each $k \in \mathcal{R}, \lambda_i^k(t)$ provides the least expected value with respect to criterion k .

A temporary vector label, $\eta_i(t) = \{ \eta_i^k(t) \}_{k \in R}$, is used in updating the labels. When assessing $\eta_i(t)$, for a given $k \in R$, if $\eta_i^k(t) < \lambda_i^k(t)$, then set $\lambda_i^k(t) = \eta_i^k(t)$ and insert node i into the SE list for subsequent scanning. A pointer $\pi_i^k(t)$ is associated with each $\lambda_i^k(t)$ to indicate the successor node from node i , at time $t \in S$ with respect to criterion k . The pointers are used to reconstruct the APPS upon termination of the algorithm. The description of the APPS algorithm, procedural steps and associated proofs are provided hereafter. In the algorithm description, $\Gamma^{-1}(j)$ denotes the predecessor nodes of node j , i.e. $(i: (i,j) \in \mathcal{A})$.

Algorithm APPS

Step 0 (Initialization):

Initialize the label vectors.

$$\lambda_i^k(t) = \infty, \forall i \in \mathcal{V} \setminus N, k \in R, t \in S.$$

$$\pi_i^k(t) = \infty, \forall i \in \mathcal{V} \setminus N, k \in R, t \in S.$$

$$\lambda_N^k(t) = 0, \forall k \in R, k \in R, t \in S.$$

$$\pi_N^k(t) = N, \forall k \in R, t \in S.$$

Create the SE list and insert the destination node N into the SE list.

Step 1 (Select Node for Scanning):

If the SE list is empty, go to step 3. Otherwise, select and delete the first node from the SE list. Call this node the current node j .

Step 2 (Update the Node Labels):

For each $i \in \Gamma^{-1}(j)$,

For each $t \in \mathcal{S}$,

For each $k \in R$,

compute temporary label values $\eta_i^k(t)$, where

$$\eta_i^1(t) = \sum_{z_1=1}^D [c_{ij}^{1z_1}(t) + \lambda_j^1(t + c_{ij}^{1z_1}(t))] \cdot \rho_{ij}^{1z_1}(t)$$

$$\eta_i^g(t) = \sum_{z_g=1}^D c_{ij}^{gz_g}(t) \cdot \rho_{ij}^{gz_g}(t) + \sum_{z_1=1}^D \lambda_j^g(t + c_{ij}^{1z_1}(t)) \cdot \rho_{ij}^{1z_1}(t), \forall g \in R \setminus 1.$$

If $\eta_i^k(t) < \lambda_i^k(t)$, then $\lambda_i^k(t) = \eta_i^k(t)$, $\pi_i^k(t) = j$ and $SE = SE \cup \{i\}$.

If all $i \in \Gamma^{-1}(j)$ have been considered, return to step 1.

Step 3 (Termination):

Stop.

The algorithm terminates with the adaptive path strategies that provide the least expected value for each criterion from all origins to a select destination for each departure time in the peak period. It is worth noting that a more efficient method is proposed here (in step 2) for computing the expected value of non-travel time criteria, $\eta_i^g(t) \forall g \in R \setminus 1$, than was given in Miller-Hooks and Mahmassani (2000).

Proposition 3.1. Upon termination, the APPS algorithm generates the adaptive path strategies that provide the least expected value for each criterion.

Proof. Upon termination, $\lambda_i^k(t)$ is the label associated with the hyperpath for node i

and departure time t that has the least expected value with respect to criterion k . No other hyperpath with associated label, $\psi_i^k(t)$, exists such that $\psi_i^k(t) < \lambda_i^k(t)$. If such a hyperpath does exist, then one of the following two conditions must be possible. (1) $\psi_i^k(t)$ was eliminated by another label or (2) $\psi_i^k(t)$ was never considered in step 2 of the algorithm. If (1) is true, then this other label would also lead to the elimination of the hyperpath associated with $\lambda_i^k(t)$, a contradiction. Thus, $\psi_i^k(t)$ must never have been considered – condition (2). If (2) holds, then either $\psi_i^k(t)$ contains a subhyperpath that was eliminated and thus, the hyperpath associated with $\psi_i^k(t)$ must be eliminated as it is suboptimal, or the SE list is not empty, contradicting the assumption of termination. The same argument holds for each criterion, establishing that all hyperpaths associated with the labels upon termination of the algorithm have the least expected value for the corresponding criteria. ♦

Proposition 3.2. In the worst-case, the algorithm has computational complexity of $\sim O(V^3 \cdot I^2 \cdot P \cdot R)$, where $V = |\mathcal{V}|$ is the number of nodes in the network, $I = |\mathcal{S}|$ is the number of time intervals within the period of interest, $P = |\mathcal{P}|$ is the maximum number of possible arc weights for a given criterion and $R = |\mathcal{R}|$ is the number of criteria under consideration.

Discussion The worst-case computational complexity of this algorithm is similar to that of the ELB algorithm. There are at most $(V-1)^2 \cdot I$ repetitions of step 2. However, in this algorithm, step 2 requires at most $(V-1) \cdot I \cdot P \cdot R$ in place of $(V-1) \cdot I \cdot P$ computations. The resulting worst-case computational complexity is $(V-1)^2 \cdot I \cdot (V-$

1)·I·P·R or $\sim O(V^3 \cdot I^2 \cdot P \cdot R)$. ♦

Note that the worst-case computational complexity of the ELB algorithm is $\sim O(V^3 \cdot I^2 \cdot P)$. That is, the APPS algorithm has the same worst-case computational complexity as that of the ELB algorithm performed R times. However, the APPS algorithm is superior to the ELB algorithm in terms of the average computational performance, which is examined through numerical experiments discussed in Section 3.5.

3.4 ILLUSTRATIVE EXAMPLE PROBLEM

This section is designed to illustrate the essential steps of the APPS algorithm. A simple example problem shown in Figure 3.3 is used for this demonstration.

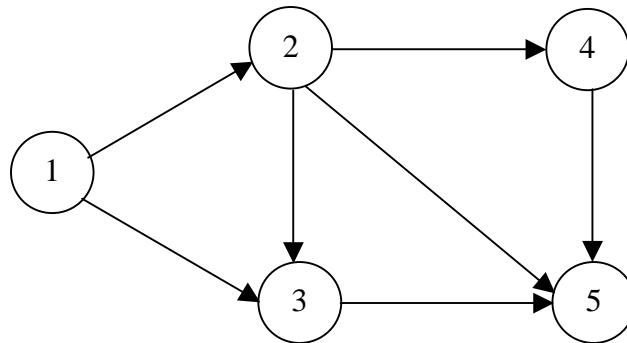


Figure 3.3. Illustrative example.

Table 3.1. Probabilistic time and cost data.

Time	arc(1,2)	arc(1,3)	arc(2,3)	arc(2,4)	arc(2,5)	arc(3,5)	arc(4,5)
$t=0$	1(0.8) 2(0.2)	2(1.0)	1(0.9) 3(0.1)	3(0.9) 4(0.1)	2(0.8) 6(0.2)	3(0.2) 5(0.8)	1(0.8) 2(0.2)
$t=1$	2(0.3) 4(0.7)	1(0.8) 2(0.2)	2(0.8) 3(0.2)	2(0.8) 4(0.2)	5(0.9) 7(0.1)	2(0.5) 4(0.5)	1(0.9) 3(0.1)
$t=2$	1(0.8) 2(0.2)	3(0.3) 4(0.7)	1(0.9) 3(0.1)	1(0.3) 2(0.7)	3(0.2) 6(0.8)	1(0.8) 2(0.2)	1(0.9) 2(0.1)
$t=3$	1(0.5) 3(0.5)	1(0.8) 2(0.2)	3(0.5) 4(0.5)	3(0.3) 4(0.7)	7(0.5) 8(0.5)	3(0.5) 4(0.5)	4(0.3) 7(0.7)
$t=4$	2(0.8) 3(0.2)	2(1.0)	2(0.8) 4(0.2)	1(0.5) 3(0.5)	6(0.9) 7(0.1)	5(0.8) 6(0.2)	5(0.5) 6(0.5)
Cost							
$t=0$	3(0.8) 4(0.2)	7(0.9) 9(0.1)	3(0.8) 4(0.2)	1(0.2) 3(0.8)	7(0.3) 8(0.7)	8(0.2) 9(0.8)	4(0.8) 5(0.2)
$t=1$	4(0.5) 5(0.5)	10(0.9) 11(0.1)	4(0.5) 5(0.5)	2(0.9) 4(0.1)	6(0.5) 8(0.5)	7(0.3) 8(0.7)	5(0.9) 6(0.1)
$t=2$	2(0.8) 3(0.2)	7(0.8) 9(0.2)	3(0.9) 4(0.1)	2(0.9) 4(0.1)	6(0.5) 8(0.5)	7(0.3) 8(0.7)	3(0.4) 7(0.6)
$t=3$	3(0.8) 4(0.2)	5(0.1) 7(0.9)	3(0.8) 4(0.2)	2(0.5) 3(0.5)	7(0.3) 8(0.7)	5(0.9) 6(0.1)	2(0.1) 4(0.9)
$t=4$	3(0.9) 4(0.1)	8(0.5) 9(0.5)	3(0.9) 5(0.1)	2(0.5) 3(0.5)	6(0.5) 8(0.5)	4(0.4) 8(0.6)	7(0.3) 8(0.7)

*arc attribute (associated probability)

Assume there are two criteria to be considered: travel time and travel cost. These criteria are assigned to criteria 1 and 2, respectively. The period of interest is discretized into five time intervals, $S=\{0,\dots,4\}$. The associated data of the example network are provided in Table 3.1 in the form of the possible travel time/cost with associated probability of occurrence given parenthetically. Due to the repetitive process of the computations, only a portion of the entire process for determining the APPS from every node to the destination (node 5) for each departure time in the peak period is presented.

Initialize the elements of the vector labels. $SE = \{5\}$.

Iteration 1

Scan node 5. $SE = \{ \}$.

$j = 5, i \in \{2, 3, 4\}$.

For $i = 2$,

$t = 0$,

$k = 1$,

$$\begin{aligned} \eta_2^1(0) &= \sum_{z_1=1}^2 [c_{25}^{1z_1}(0) + \lambda_5^1(0 + c_{25}^{1z_1}(0))] \cdot \rho_{25}^{1z_1}(0) \\ &= (2+0) \cdot 0.8 + (6+0) \cdot 0.2 = 2.8. \end{aligned}$$

Set $\lambda_2^1(0) = 2.8, \pi_2^1(0) = 5$ and $SE = \{2\}$.

$k = 2$,

$$\begin{aligned} \eta_2^2(0) &= \sum_{z_2=1}^2 c_{25}^{2z_2}(0) \cdot \rho_{25}^{2z_2}(0) + \sum_{z_1=1}^2 \lambda_5^2(0 + c_{25}^{1z_1}(0)) \cdot \rho_{25}^{1z_1}(0) \\ &= (7 \cdot 0.3 + 8 \cdot 0.7) + (0 \cdot 0.8 + 0 \cdot 0.2) = 7.7. \end{aligned}$$

Set $\lambda_2^2(0) = 7.7, \pi_2^2(0) = 5$ and $SE = \{2\}$.

-
- (Continue to loop over t .)
-

For $i = 3$ and 4 , each $t \in S$ and each $k \in R$, compute the label components and associated pointers. Make necessary updates. Within this iteration, at least one component of each of the labels at nodes 2, 3 and 4 has been updated. Thus, $SE = \{2,3,4\}$ at the end of this iteration. Figure 3.4 shows $\lambda_i^k(t)$ and $\pi_i^k(t), \forall k \in R$ associated with nodes 2, 3, and 4, respectively, at the end of this iteration. The values are presented in the form of $(\lambda_i^k(t), \pi_i^k(t))$.

Time	Node 2		Time	Node 3		Time	Node 4	
	k=1	k=2		k=1	k=2		k=1	k=2
0	(2.8, 5)	(7.7, 5)	0	(4.6, 5)	(8.8, 5)	0	(1.2, 5)	(4.2, 5)
1	(5.2, 5)	(7.0, 5)	1	(3.0, 5)	(7.7, 5)	1	(1.2, 5)	(5.1, 5)
2	(5.4, 5)	(7.0, 5)	2	(1.2, 5)	(7.7, 5)	2	(1.1, 5)	(5.4, 5)
3	(7.5, 5)	(7.7, 5)	3	(3.5, 5)	(5.1, 5)	3	(6.1, 5)	(3.8, 5)
4	(6.1, 5)	(7.0, 5)	4	(5.2, 5)	(6.4, 5)	4	(5.5, 5)	(7.7, 5)

(a)

(b)

(c)

Figure 3.4. Solutions for iteration 1.Iteration 2

Scan node 2. SE = {3,4}.

 $j = 2, i \in \{1\}$.For $i = 1$, $t=0$ $k=1$,

$$\begin{aligned} \eta_1^1(0) &= [c_{12}^{11}(0) + \lambda_2^1(0 + c_{12}^{11}(0))] \cdot \rho_{12}^{11}(0) + [c_{12}^{12}(0) + \lambda_2^1(0 + c_{12}^{12}(0))] \cdot \rho_{12}^{12}(0) \\ &= (1+5.2) \cdot 0.8 + (2+5.4) \cdot 0.2 = 6.44. \end{aligned}$$

Set $\lambda_1^1(0) = 6.44$, $\pi_1^1(0) = 2$ and SE = {3,4,1}.

-
- (Continue to loop over k and t .)
-

Continue in the same manner until the SE list is empty when step 1 is called.

The final APPS for every node and departure time in the peak period are provided in Figure 3.5.

Time	Node 1	
	$k=1$	$k=2$
0	(3.2, 3)	(10.0, 2)
1	(2.9, 3)	(11.7, 2)
2	(8.4, 2)	(9.8, 2)
3	(6.4, 3)	(10.2, 2)
4	(7.2, 3)	(10.1, 2)

(a)

Time	Node 2	
	$k=1$	$k=2$
0	(2.8, 5)	(6.8, 4)
1	(5.2, 5)	(6.8, 4)
2	(4.9, 3)	(7.0, 5)
3	(7.5, 5)	(7.7, 5)
4	(6.1, 5)	(7.0, 5)

(b)

Time	Node 3	
	$k=1$	$k=2$
0	(4.6, 5)	(8.8, 5)
1	(3.0, 5)	(7.7, 5)
2	(1.2, 5)	(7.7, 5)
3	(3.5, 5)	(5.1, 5)
4	(5.2, 5)	(6.4, 5)

(c)

Time	Node 4	
	$k=1$	$k=2$
0	(1.2, 5)	(4.2, 5)
1	(1.2, 5)	(5.1, 5)
2	(1.1, 5)	(5.4, 5)
3	(6.1, 5)	(3.8, 5)
4	(5.5, 5)	(7.7, 5)

(d)

Figure 3.5. APPS solutions.

As depicted in Figure 3.5, upon termination of the algorithm, $\lambda_1^1(0) = 3.2$, $\lambda_1^2(0) = 10.0$ with associated hyperpath pointers, $\pi_1^1(0) = 3$ and $\pi_1^2(0) = 2$, respectively. The APPS from node 1 at departure time $t = 0$ indicate that the traveler should head to either node 2 for the least expected travel cost or node 3 for the least expected travel time. If the traveler chooses to follow the instructions associated with travel time and moves to node 3, (s)he will be instructed to continue directly to node 5. On the other hand, if the driver chooses to depart from node 1 to go to node 2 and the arrival time at this node is $t=1$, two efficient moves are suggested depending on the traveler's preference at that point in time. If the traveler wishes to obtain the least expected travel time, the best option is to go to node 5 directly. Otherwise, heading to node 4 prior to reaching node 5 will result in the least expected travel cost. For arrival time $t = 2$ at node 2, going to node 5 directly is suggested for the least expected travel

cost and going to node 3 prior to reaching node 5 is suggested for the least expected travel time. The APPS from node 1 to node 5 departing from node 1 at $t = 0$ are portrayed in Figure 3.6.

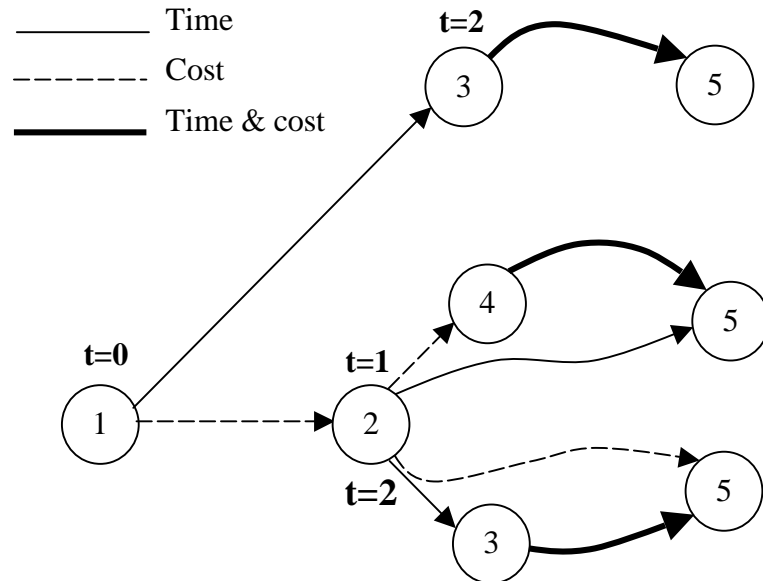


Figure 3.6. APPS from node 1 to node 5.

3.5 NUMERICAL EXPERIMENTS

This section presents the results from numerical experiments conducted on randomly generated networks with randomly generated time-varying PMFs of the arc attribute variables. The objective is to examine improvements in average computational time attained by the use of the APPS algorithm over the ELB algorithm performed repeatedly, once for each criterion. The average run time of the APPS algorithm was compared to that of the ELB algorithm through experiments on networks with either 100, 500 or 1000 nodes, a peak period of 120 time intervals, 5 elements in each PMF and three levels of the number of criteria (2, 4 and 6).

3.5.1 Experimental Design

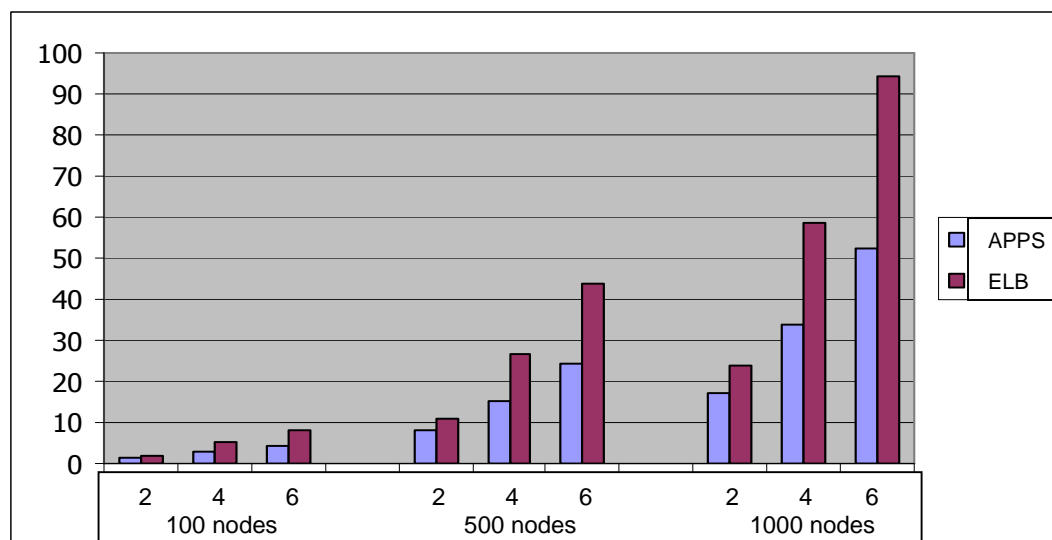
The computational experiments were conducted on a DEC Alpha XP1000 professional workstation with 1 gigabyte ram and 2 gigabyte swap, running Digital 4.0E operating system, using Digital's C++ compiler. The number of nodes (V), number of time intervals (I), number of elements in the PMFs (P) and number of criteria (R) are predesignated. In accordance with typical traffic networks, the average in-degree and out-degree are each four, ranging between 2 and 9. The same methodology used for creating the STV networks as described in Miller-Hooks and Mahmassani (2000) is extended for use in MSTV networks and is applied here. Each criterion was assumed to be uniformly distributed with fixed lower bound of one unit. The upper bound was permitted to linearly increase from 5 to 10 units in the first half of the peak period and then linearly decrease to 5 units in the second half of the peak period. A FIFO SE list was employed in the APPS algorithm. For each test (i.e. each network size and chosen number of criteria), 30 runs were completed, corresponding to 30 randomly selected destinations. The average of these 30 runs is reported.

3.5.2 Results and Discussion

Computational savings achieved by the use of the APPS algorithm over the ELB algorithm were evaluated through experiments on several network configurations. Average run times in c.p.u. seconds over 30 destinations for both algorithms are displayed in Table 3.2.

Table 3.2. Average run time comparisons.

V	R = 2		R = 4		R = 6	
	APPS/ELB	% Improve	APPS/ELB	% Improve	APPS/ELB	% Improve
100	1.48/2.06	28.16	2.93/5.03	41.75	4.35/8.03	45.83
500	8.30/11.03	24.75	15.45/26.79	42.33	24.38/43.88	44.44
1000	17.03/24.02	29.10	33.63/58.41	42.42	52.30/94.35	44.57

**Figure 3.7. APPS/ELB run time comparisons.**

The results show that the APPS algorithm improves the average run time approximately 25-30% over the ELB algorithm for the instances with two criteria. The improvements increase with the number of criteria. For instance, the percentage of improvement reaches 45% on average for the instances with six criteria. Thus, the APPS algorithm provides significant savings in computational effort over the repetitive execution of the ELB algorithm for obtaining the same solutions, especially when the number of considered criteria is large. Such improvements stem directly from the efficiency of handling more than one criterion in Step 2 of the APPS algorithm. This efficiency plays a major role in diminishing the average run time.

Furthermore, the APPS algorithm performed much better than its worst-case computational complexity, $\sim O(V^3 \cdot I^2 \cdot P \cdot R)$. For example, the average run time for the 1,000 node network with five elements in the PMFs, 120 time intervals, and two criteria is 17.03 c.p.u. seconds, which is considerably less than one thousand times the average run time of the 100 node network with, 120 time intervals and two criteria (requiring 1.48 c.p.u. seconds).

3.6 CONCLUSIONS

In this chapter, the concept of adjustable preference path strategies (APPS) is introduced. APPS are defined as path strategies that enable a traveler to adaptively select the best path in accordance with the traveler's changing preferences and revealed arrival times at intermediate locations. Such a solution strategy permits a traveler to alter his/her preferred criterion at each node en route to the destination and is of importance in providing on-line path finding assistance in traffic networks.

The APPS algorithm is developed for determining APPS in MSTV transportation networks. Specifically, adaptive path strategies that seek to minimize the expected value of each of multiple criteria are generated from all origins to a designated destination for all departure times in a period of interest. Although identical solutions can be attained by performing the ELB algorithm (2000) multiple times, once for each criterion, the APPS algorithm offers significant computational savings as indicated by the results of numerous numerical experiments.

CHAPTER 4

THE SAFEST ESCAPE PROBLEM

4.1 INTRODUCTION

In this chapter, an exact algorithm, the SEscape (Safest Escape) algorithm, is proposed for determining the optimal set of *a priori* path flows in dynamic networks with time-varying arc traversal times and stochastic, time-varying (STV) arc capacities. The SEscape algorithm seeks the pattern of flow that maximizes the minimum path probability of successful arrival at the sink of supply originating at multiple source nodes. That is, for a given flow pattern, the probability of successful arrival by each unit of flow at the sink along each constituent path is assessed. The path with the minimum success probability is identified. The flow pattern whose minimum success probability path has the maximum success probability is optimal. The problem of determining this pattern of flow is referred to as the SEscape problem. The concept is illustrated in Figure 4.1, where the minimum success probability paths in a static network are shown in bold.

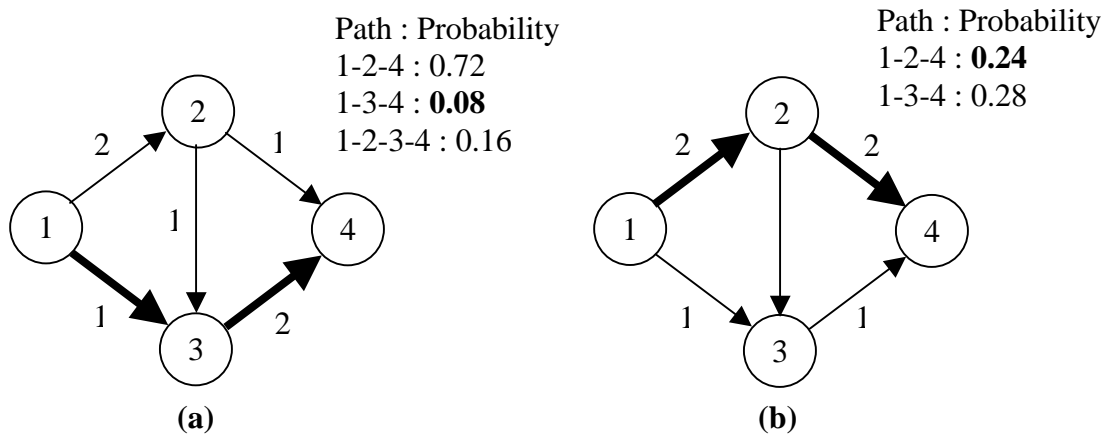


Figure 4.1. SEscape problem.

Suppose two possible patterns of flow exist for shipping three supply units from node 1 to node 4 as depicted in Figure 4.1. The flow pattern shown in Figure 4.1b is optimal for the SEscape problem since the associated minimum success probability (0.24 on path 1-2-4) is greater than that of the flow pattern shown in Figure 4.1a (0.08 on path 1-3-4).

The SEscape algorithm was developed to provide egress instructions to evacuees in the event that rapid evacuation of a large damaged building, e.g. a burning building or a building that has come under attack by enemy or natural catastrophe, is required. The concept of choosing the solution whose constituent paths contain the maximum minimum probability of success (the SEscape problem) is similar conceptually to the problem of choosing the solution with the minimum time until the last evacuee escapes, i.e. the solution that leads to the minimum evacuation time. In the SEscape problem, the risk incurred by any person who is forced to take the greatest risk is minimized (i.e. their probability of successful escape is maximized).

In a network representation of the building evacuation problem, the network represents the layout of the circulation systems of the building, where nodes correspond to locations inside the building (such as offices, meeting rooms, lobbies, lavatories, and building exits) and arcs correspond to the passageways that connect these locations (such as staircases, elevator shafts, doorways, corridors and ramps). The supply units represent the people to be evacuated. The nodes at which the evacuees are located when the evacuation begins are called source nodes and the node corresponding to building exit locations are referred to as sink nodes. The arc travel

time represents the amount of time required for traversing the arc. The capacity of an arc is the number of people that can pass through the associated arc per unit of time. The arc capacities are dependent upon the size and type of corresponding passageway.

Emergency evacuations are often characterized by dangers that strengthen and spread over time. When a large number of people must be evacuated from the building simultaneously, issues concerning capacity of the arcs arise. Circumstances in an evacuation induce the possibility that successful egress may be inhibited by partial or complete failure of key escape paths. Moreover, one cannot know how the situation will progress with certainty even if the exact location and type of event that initiated the need for the evacuation is known. Thus, in determining the optimal instructions, it is important to explicitly consider the time-varying and uncertain nature of capacities inherent in such circumstances. Instructions that do not consider the evolution of damage over time and threats of probable additional destruction and deterioration can result in suboptimal decisions that can lead to unnecessary imposed risk and unnecessary lost lives (Miller-Hooks and Krauthammer, 2002).

To represent such emergency evacuation conditions, in this work, a dynamic network flow problem with time-varying arc traversal times and STV arc capacities is considered. In dynamic networks, flow moves through the network over time and arc capacities are recaptured over time. Conventionally, network attributes (arc travel times, arc capacities and supply) in such dynamic networks are assumed to be time-invariant. In many real-world applications, however, network attributes fluctuate over time, and thus, it is critical that the inherent time-varying nature of the network

attributes is explicitly considered. The SEscape problem is considered in such a dynamic, time-varying environment, as depicted in Figure 4.2. In the figure, the period of interest is assumed to be discretized into three time intervals ($t=0, 1$ and 2) and arc travel times are known deterministically. Arc capacities are discrete random variables with probability distributions that vary with time. The possible capacities and associated probabilities of occurrence are given. For instance, four possible capacities on arc (1,2) when departing from node 1 at $t=0$ are 0 with probability 0.4, 1 with probability 0.2, 2 with probability 0.1, and 4 with probability 0.3. These values are shown in the figure in decreasing order from top to bottom.

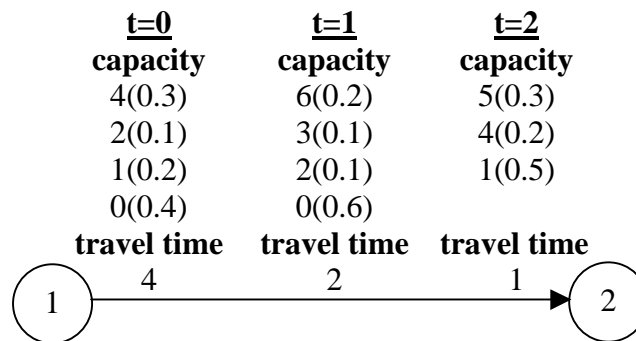


Figure 4.2. Time-varying travel times and STV capacities.

4.2 LITERATURE REVIEW

Existing approaches proposed in the literature to solve related network flow problems generally do not explicitly model the variable and uncertain conditions inherent in circumstances warranting emergency evacuation. A host of researchers have addressed network flow problems in static networks (see Ahuja et al. (1993) for additional detail on several network flow problems). One of the most studied network flow problems is the minimum cost flow problem, in which the aim is to ship all

supply units from the source to the sink with minimum total cost. Yamada (1996) implemented a classical minimum cost flow algorithm to evaluate the evacuation plan for Yokosuka City, Japan. Calvete (2003) presented modifications to the well-known network simplex method to solve the minimum cost flow problem with side constraints. The side constraints require each arc in a specified subset to carry the same amount of flow. For the sake of improving the computational complexity, Orlin (1993) and Sokkalingam et al. (2000) developed polynomial time algorithms for solving the same problem. Ahuja et al. (2002) proposed a new pivot rule in the network simplex algorithm for solving the minimum cost flow problem, which requires at most k ($k \leq$ number of nodes in the network) consecutive degenerate pivots.

Several other authors considered the minimum cost flow problem with multiple objectives. Noda and Matin (2001) addressed the bi-objective minimum cost flow problem with integral flow variables. The proposed algorithm consists of two stages. The first stage generates all integer solutions on the efficient boundary in the objective space by using a modified network simplex method. Given these solutions, the second stage identifies the non-dominated solutions that do not lie on the efficient boundary without generating any dominated solution. Cova and Johnson (2003) formulated a simple linear program based on the conventional minimum cost flow problem to address the problem of finding optimal lane-based evacuation routing plans. The constraint method for multi-objective programming was employed. The primary objective is to minimize total distance and the secondary objectives, minimizing vehicle merging-conflicts and preventing crossing-conflicts at

intersections, are set as constraints. Other interesting works that address network flow problems in static environments include Aggarwal et al. (1998), Pióro et al. (2002), and Curet et al. (2002).

Numerous works incorporate the dynamic nature of network attributes that is present in many real world applications. A survey of algorithms, applications and implementations of several dynamic network flow problems is provided by Aronson (1989). Ford and Fulkerson (1962) studied the maximum dynamic flow problem whose objective is to ship as much flow as possible from the source to the sink in a given period of time. They proposed a technique employing a minimum cost flow algorithm as a subroutine to find maximum flow. Sparked by their approach, several authors have addressed a generalization of the maximum flow problem, the universal maximal dynamic flow problem (Minieka, 1971; Wilkinson, 1971; Halpern, 1979; and Fleischer, 2001). The objective is to determine a pattern of flow that maximizes the amount of flow arriving at the sink for every time interval within the considered time bound. Anderson and Philpott (1994), and Orda and Rom (1995) dealt with both original and generalized maximum flow problems with a continuous representation of time. Nagy and Akl (2003) studied the maximum flow problem in a real-time setting. They proposed solution approaches for recomputing a new maximum flow without starting from scratch when real-time information regarding the network structure or capacities is received.

Closely related to the maximum dynamic flow problem, the quickest flow problem is concerned with determination of a flow pattern for shipping supply units from a source to a sink such that the time at which the last unit arrives at the sink is

minimized. Burkard et al. (1993) conjectured a relationship between these two flow problems and proposed several polynomial algorithms and a strongly polynomial algorithm for solving the quickest flow problem. Chalmet et al. (1982), and Hamacher and Tufekci (1997) considered an extension of the quickest flow problem to multiple sources, the evacuation problem. Hoppe and Tardos (2000) presented the first polynomial-time algorithm for the quickest transshipment problem, which extends the quickest flow problem to multiple sources and multiple sinks. Chen and Chin (1990), Rosen et al. (1991), and Calvete (2004) studied a variant of the quickest flow problem: the quickest path problem. The quickest path problem seeks a single path for shipping the supply from a source to a sink with minimum total traversal time, where the traversal time of an arc depends on the rate of flow on that arc. Jarvis and Ratliff (1982) considered dynamic network flow problems with three different objectives: 1) maximize flow in the first t time intervals, for every t (i.e. the universal maximal dynamic flow problem), 2) minimize the average time and 3) minimize the time the last unit arrives at the sink (i.e. the quickest flow problem). They showed that the optimal solution for one objective is also optimal for the other two and suggested that a standard minimum cost flow algorithm could be used to solve all three problems.

In conjunction with building evacuation, Choi et al. (1988) considered three related dynamic network flow problems (i.e. the maximum flow, minimum cost and quickest flow problems) with additional side constraints. The side constraints require that the capacity of an arc is a function of the rate of flow on the incoming arcs. Francis (1981) presented a uniformity principle for building evacuation, requiring that

if the building is to be evacuated in minimum time, evacuees will be assigned to a set of routes having the identical evacuation time. All of the aforementioned works model time discretely. Fleischer and Tardos (1998) extended several algorithms for dealing with discrete-time dynamic flow problems to solve their continuous-time counterparts.

While many papers have addressed network flow problems in dynamic (but time-invariant) networks, few works have considered the time-dependent characteristics of network attributes (i.e. arc traversal times, capacities and supply at the source). Miller-Hooks and Stock Patterson (2004) proposed the Time-Dependent Quickest Flow Problem (TDQFP) algorithm for the quickest evacuation problem in time-varying capacitated networks with dynamic flow characteristics. Cai et al. (2001a) developed solution algorithms to solve the time-varying minimum cost flow problem with three waiting policies at the nodes. Cai et al. (2001b) also dealt with the maximum flow and universal maximum flow problems in such time-varying instances.

Works discussed thus far consider only deterministic network flow problems. Several researchers have addressed stochastic flow problems in the form of network connectivity and reliability, where the nodes or arcs may randomly fail with known probability. Frank and Gaul (1982) considered two probabilities of connectedness: 1) the probability that the entire network is connected and 2) the probability that two selected nodes are connected. They presented bounds and approximations to these two problem classes. Lucet and Manouvrier (1999) investigated several methods for

finding the probability that a certain subset of nodes is connected. These works do not take into account flow through the network.

System reliability is defined as the probability that the network can accommodate a particular level of flow. Jentsch (1998) addressed the problem of computing system reliability in special network topologies. Only two arc states are considered: zero or full capacity. Lin (2001, 2002a) proposed an exact method to assess the system reliability in static networks with stochastic arc capacities. He further extended the method for solving the system reliability problem with two-commodities (Lin, 2002b). Lin (2003) proposed an additional concept of system reliability based on the quickest path problem. Given stochastic arc capacities, the minimum time for shipping supply units on a single path depends on the realization of the arc capacities. For each realization, the probability of occurrence and the corresponding quickest path for shipping supply units are identified. System reliability in this work is defined as the probability that the supply units can be shipped from the source to the sink within a time bound.

Another approach for examining the performance of capacitated networks with random arc failures is to consider the expected value of maximum flow. Because computing such an expected value is NP-hard (as mentioned in, for example, Nagamochi and Ibaraki, 1991), upper and lower bounds on the expected maximum flow is used as an approximation to the exact value. Carey and Hendrickson (1984) developed several algorithms for computing bounds on the expected maximum flow. They showed that an upper bound to the expected maximum flow can be computed by solving the maximum flow problem on the network with arc capacities set to their

expected values. Based on an algorithm given in the previous work, Nagamochi and Ibaraki (1991) proposed necessary and sufficient conditions for examining whether the lower bound equals the exact value of the expected maximum flow. All of these works are concerned with the evaluation of the network performance, rather than providing routing plans. Furthermore, they do not consider the time-dependency of the network attributes.

Other works have addressed stochastic network flow problems where routing plans are determined. By assuming time-invariant network attributes, Talebi and MacGregor Smith (1985) modeled the stochastic evacuation problem with analytical queuing network models. The expected evacuation time is used as the performance measure in optimal egress analysis. Glockner et al. (2000, 2001) presented a multistage stochastic linear programming formulation for the minimum cost flow problem in dynamic networks with uncertain arc capacities. Travel times are assumed to be deterministic and time-invariant. Decomposition techniques were developed for use in the multistage linear program. Karbowicz and MacGregor Smith (1984) proposed a simulation-based methodology for determining the evacuation paths along which to send evacuees. The proposed approach employs a K -shortest paths algorithm to identify the K shortest paths from every source to every sink. Then, occupant egress is simulated along the set of the first shortest paths to detect queueing. If queueing exists, a portion of occupants who experience delay is rerouted and queueing is evaluated through a simulation. This iterative process continues until the combination of paths that minimizes total evacuation time is obtained. Because the proposed procedure involves a number of computations and simulation runs, it is

expected to perform poorly when significant queueing is found. It appears that no exact algorithm has been proposed for efficiently determining optimal evacuation paths, where the uncertain, dynamic and time-varying conditions inherent in emergency circumstances are explicitly considered.

The primary contribution of this chapter is the development of the methodological steps for providing optimal *a priori* (safest escape) instructions for egress in emergency evacuation. The dynamic, time-varying and uncertain nature of passageway capacities inherent in emergency incidents warranting evacuation is explicitly considered. In the next section, several concepts and related performance measures that one may consider in determining evacuation instructions are discussed.

4.3 CONCEPTUAL FRAMEWORK

The goal of providing evacuation plans in emergency incidents is to maximize the safety of evacuees. Conventionally, evacuation time (i.e. the time until the last person exits) and total time for evacuation have been used as a primary criterion in the selection of egress paths (Chalmet et al., 1982; Karbowicz and Macgregor Smith, 1984; Talebi and MacGregor Smith, 1985; Choi et al., 1988; Hamacher and Tufekci, 1997; Fleischer and Tardos, 1998; Cai et al., 2001a; Miller-Hooks and Stock Patterson, 2004). Time is used as a surrogate for risk and the set of path flows that yields the minimum time required for evacuating all occupants from a disaster area is preferred. Such approaches are useful when capacities of the passageways are known deterministically. In emergency incidents, however, how the situation will progress is uncertain, and thus, one cannot know *a priori* the number of people who will be able

to successfully pass through a given passageway at any point in time with certainty. There may be some probability that successful egress along one or more passageways will be inhibited. That is, capacities can be known *a priori* only probabilistically. Such probabilities are useful for developing evacuation strategies in stochastic environments. For example, one would prefer an evacuation path with long journey time, but high likelihood of successful escape to a path with short journey time, but very low probability of successful escape.

Miller-Hooks and Krauthammer (2002) propose an intelligent evacuation, rescue and recovery (IERR) concept for large buildings that would enable prediction in real-time of future arc capacities and times including their distributions. With the advent of new techniques and the development of concepts that exploit information that can be retrieved via these technologies, the information required to derive the probability of successful escape as described herein is becoming a reality.

In this section, several criteria for evaluating flow patterns in networks with stochastic arc capacities are discussed. Specifically, capacities are modeled herein as discrete random variables with probability distribution functions that vary with time. The arc attributes are assumed to be independent over space and time and the period of interest is considered discretely.

4.3.1 Expectation and Path Flows

Stochastic problems are frequently addressed by considering the expected values of the random characteristics. By replacing the random variables by their expected values, the stochastic problem is transformed into a deterministic one, and thus, the

difficulties that stem from working with random variables are eliminated. In networks, where arc capacities are random quantities, there may be more than one random quantity for which the expectation is useful. For example, one might consider the expected capacity along each arc or the expected number of flow units that can successfully cross each arc (referred to herein as the expected flow). To illustrate these concepts, the example with discrete random variables and discrete time given in Figure 4.2 is revisited.

In the example, the expected capacity of arc (1,2) at $t = 0$ can be computed by

$$\sum_{n=0}^4 n \cdot P\{u_{12}(0) = n\} = 0 \cdot (0.4) + 1 \cdot (0.2) + 2 \cdot (0.1) + 3 \cdot (0) + 4 \cdot (0.3) = 1.6, \text{ where}$$

$P\{u_{ij}(t) = n\}$ represents the probability that the capacity of arc (i,j) at time t is equal to n . If one replaces the random arc capacities by their expected values, the resulting information that can be obtained may not be useful in the context of providing evacuation instructions. For instance, for any chosen path, one will only know the expected number of units that can be shipped along the path. However, for a particular realization of the network, the actual available capacity may be far less than its expectation.

Alternatively, one might consider the expected flow. While the expected capacity of an arc at a given time interval is fixed, the expected flow along an arc depends on the number of units that are shipped across the arc. The expected flow along arc (i,j) at time t given that k units attempt to traverse the arc, can be computed by equation 4.1.

$$\sum_{n=0}^{k-1} n \cdot P\{u_{ij}(t) = n\} + k \cdot P\{u_{ij}(t) \geq k\} \quad (4.1)$$

For example, in Figure 4.2, the expected flow for shipping two units through arc (1,2) at $t = 0$ can be computed as follows: $0 \cdot \mathbf{P}\{u_{12}(0) = 0\} + 1 \cdot \mathbf{P}\{u_{12}(0) = 1\} + 2 \cdot \mathbf{P}\{u_{12}(0) \geq 2\} = 1.0$.

Lemma 4.1. The computation of the expected flow given k units attempt to traverse

arc (i,j) at time t can be completed via $\sum_{n=1}^k \mathbf{P}\{u_{ij}(t) \geq n\}$.

Proof.

$$\sum_{n=0}^{k-1} n \cdot \mathbf{P}\{u_{ij}(t) = n\} + k \cdot \mathbf{P}\{u_{ij}(t) \geq k\}$$

$$= \mathbf{P}\{u_{ij}(t) = 1\} + 2 \cdot \mathbf{P}\{u_{ij}(t) = 2\} + 3 \cdot \mathbf{P}\{u_{ij}(t) = 3\} + \dots + k \cdot \mathbf{P}\{u_{ij}(t) \geq k\}$$

$$= \mathbf{P}\{u_{ij}(t) = 1\} + \mathbf{P}\{u_{ij}(t) = 2\} + \mathbf{P}\{u_{ij}(t) = 3\} + \dots + \mathbf{P}\{u_{ij}(t) \geq k\} \quad (1)$$

$$+ \mathbf{P}\{u_{ij}(t) = 2\} + \mathbf{P}\{u_{ij}(t) = 3\} + \dots + \mathbf{P}\{u_{ij}(t) \geq k\} \quad (2)$$

$$+ \mathbf{P}\{u_{ij}(t) = 3\} + \dots + \mathbf{P}\{u_{ij}(t) \geq k\} \quad (3)$$

⋮
⋮
⋮

$$+ \mathbf{P}\{u_{ij}(t) \geq k\} \quad (k)$$

$$= \mathbf{P}\{u_{ij}(t) \geq 1\} + \mathbf{P}\{u_{ij}(t) \geq 2\} + \mathbf{P}\{u_{ij}(t) \geq 3\} + \dots + \mathbf{P}\{u_{ij}(t) \geq k\}$$

$$= \sum_{n=1}^k \mathbf{P}\{u_{ij}(t) \geq n\}$$

4.3.2 Probabilities of Successful Path Traversal

Another criterion one may consider in determining optimal path flows in stochastic capacity networks is the probability of ensuring successful arrival at the sink. Such probabilities are directly related to the arc capacity occurrence probabilities. For instance, the probability that n units can traverse arc (i,j) at time t is equal to the probability that the capacity of the arc at that particular time is greater than or equal to n , $P\{u_{ij}(t) \geq n\}$.

The static network provided in Figure 4.3 is constructed to illustrate how the probability of successful path traversal can be used to evaluate paths for completing shipments. Arc capacities and corresponding probabilities of occurrence are given parenthetically, e.g. the possible capacities of arc I are 0 with probability 0.65 and 4 with probability 0.35: $P\{u_I = 0\} = 0.65$ and $P\{u_I = 4\} = 0.35$.

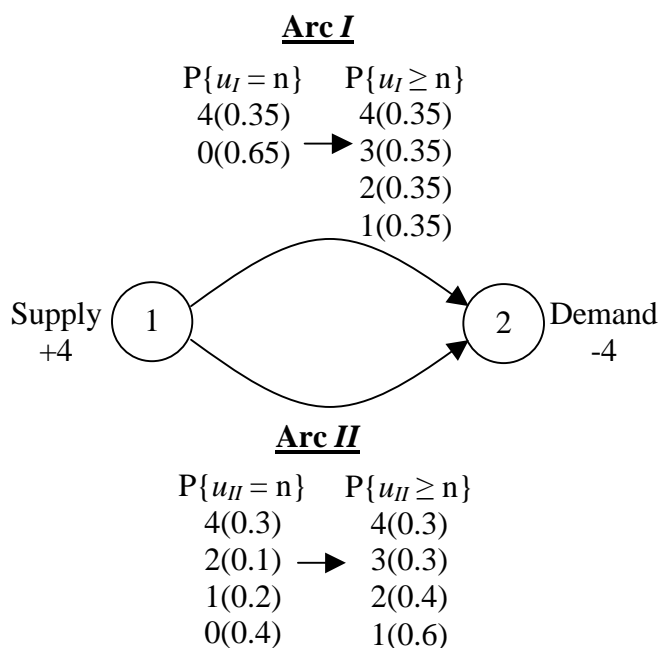


Figure 4.3. Example network.

Suppose there are four supply units at node 1. The first unit should be assigned to arc II since the probability that it will arrive at node 2 by way of this arc is greater than through arc I (i.e. $P\{u_{II} \geq 1\} = 0.6 > P\{u_I \geq 1\} = 0.35$). Given that one cannot know *a priori* whether or not the first shipment will successfully arrive at its destination, arc II is the preferred arc for shipping the second unit, because the probability that arc II can accommodate two units simultaneously, $P\{u_{II} \geq 2\}$, is greater than the probability that arc I can accommodate one unit, $P\{u_I \geq 1\}$. However, for the third unit, arc I is more attractive, because the probability that arc II can handle three units simultaneously, $P\{u_{II} \geq 3\}$, is less than the probability that arc I can handle one unit, $P\{u_I \geq 1\}$. This is also the case for shipping the last unit, where $P\{u_I \geq 2\} = 0.4 > P\{u_{II} \geq 3\} = 0.3$. Given this, the best routing plan for these four supply units is to equally split the flow across both arcs.

In Section 4.4, the relationship between the probabilities of successful path traversal and determination of the Safest Escape paths described in Section 4.1 is given.

4.4 SAFEST ESCAPE

In developing evacuation instructions in emergency events, one might consider the number of evacuees who successfully escape as a performance measure of a proposed solution (i.e. evacuation plan). It can be shown that if the evacuees follow the instructions that result from solving the maximum expected flow problem, in the long run (i.e. over many evacuations), the maximum number of lives will be saved. However, the solution to the expected flow problem may require a person to follow a

path with high likelihood of failure. That is, a single person may be asked to take exceptional risk for the good of the whole. This is portrayed in Figure 4.1. While the pattern of flow shown in Figure 4.1a has the maximum expected flow for shipping three units from node 1 to node 4, the evacuee routed on path 1-3-4 is subject to an extremely low probability of successful arrival at the sink. Albeit acceptable for many applications, the suggestion of such significantly inferior paths in life-threatening situations might raise ethical concerns.

The SEscape problem is proposed to address emergency evacuations by explicitly considering the time-varying and uncertain nature of passageway capacities inherent in such circumstances. Critical issues arising in such life-threatening situations, such as exposure to risk and fairness, are given priority. That is, rather than focusing on the system objective, the SEscape problem provides evacuation instructions such that the risk incurred by the occupant or occupants who are forced to take the greatest risk is minimized. Specifically, among the constituent paths in a pattern of flow, there is one path that has the minimum probability of successful arrival at the exit (assume no ties). The pattern of flow for which this path with the minimum success probability has the highest probability among all possible flow patterns is optimal. Figure 4.4 illustrates how the path with the minimum success probability for a given flow pattern is determined. The probabilistic arc capacities are assumed to be temporally and spatially independent.

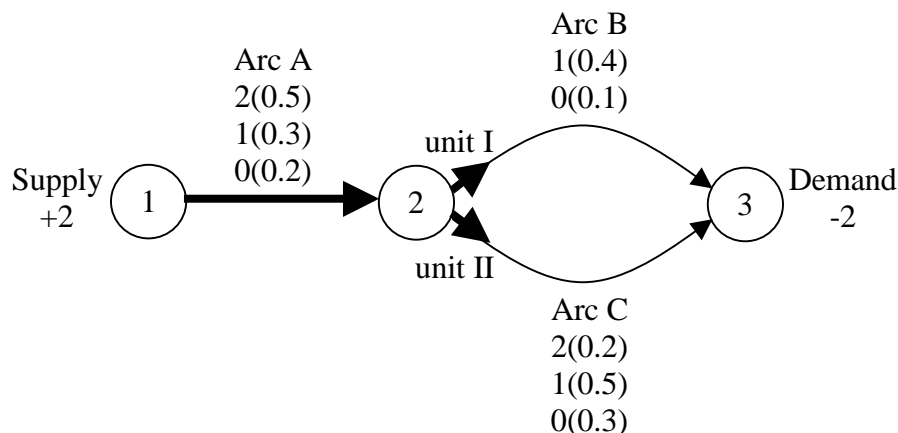


Figure 4.4. Stochastic arc capacities.

Two possible flow patterns, F_I and F_{II} , exist for shipping two supply units from node 1 to node 3: (I) split the flow equally, one along each arc as illustrated in Figure 4.4, and (II) ship both units along path A-C.

For F_I , the minimum probability that the units can successfully traverse arc A is equal to the probability that the arc can accommodate two or more units, i.e. $P\{u_A \geq 2\}$. Upon arrival at node 2, unit I is routed to arc B, where the corresponding probability of successful arc traversal is $P\{u_B \geq 1\}$. Likewise, the success probability of unit II on arc C is $P\{u_C \geq 1\}$. Therefore, for this particular flow pattern, the minimum success probability is $P\{u_A \geq 2\} \cdot P\{u_B \geq 1\} = 0.2$ on path A-B (note that path A-C has the probability of successful arrival $P\{u_A \geq 2\} \cdot P\{u_C \geq 1\} = 0.35$). The minimum success probability for F_{II} is $P\{u_A \geq 2\} \cdot P\{u_C \geq 2\} = 0.1$, computed through similar computations as shown for F_I .

One can see that for shipping two supply units from node 1 to node 3 in this network, the flow pattern F_1 maximizes the minimum path probability of successful arrival of supply at node 3. Thus, F_1 is chosen as the optimal solution with respect to the SEscape problem. This concept guarantees that the chance of successful egress of occupants who are subject to the greatest risk will be maximized. The proposed conceptual framework and specific algorithmic steps can be used in evacuation planning, enabling safer evacuation of a building in the event of military attack, fire, natural disaster, or other circumstances warranting quick escape.

4.5 NETWORK NOTATION AND PROBLEM FORMULATION

Similar notation for describing the network as used by Miller-Hooks and Stock Patterson (2004) is employed herein. Let a dynamic network $\mathcal{N} = (\mathcal{G}, U, \mathcal{B}_C, \mathcal{T})$ be a finite digraph, $\mathcal{G} = (\mathcal{V}, \mathcal{A}, \{0,1,\dots,T\})$, where \mathcal{V} is the set of nodes, \mathcal{A} is the set of directed arcs connecting the nodes, and $\{0,1,\dots,T\}$ is the time frame of interest discretized into small time intervals. The arc capacities are assumed to be discrete random variables with probability mass functions (PMFs) given by the set (U, \mathcal{B}_C) . Specifically, associated with each arc $(i,j) \in \mathcal{A}$ at time $t \in \{0,1,\dots,T\}$ is a set of D non-negative, integer-valued, time-varying capacities, $U = \{u_{ij}^z(t)\}_{(i,j) \in \mathcal{A}, t \in \{0,1,\dots,T\}, z=1,\dots,D}$, with corresponding probabilities of occurrence, $\mathcal{B}_C = \{\beta_{ij}^z(t)\}_{(i,j) \in \mathcal{A}, t \in \{0,1,\dots,T\}, z=1,\dots,D}$, and a set of non-negative real-valued time-varying travel times, $\mathcal{T} = \{\tau_{ij}(t)\}_{(i,j) \in \mathcal{A}, t \in \{0,1,\dots,T\}}$. The flow on arc $(i,j) \in \mathcal{A}$ that leaves node i at departure time $t \in \{0,1,\dots,T\}$ and arrive at node j at time $t + \tau_{ij}(t)$ is represented by $x_{ij}(t)$. $P_{ij}^n(t)$

denotes the probability that the capacity of arc (i,j) at time t is not less than n , i.e. $P\{u_{ij}(t) \geq n\}$ for $(i,j) \in \mathcal{A}$, $t \in \{0,1,\dots,T\}$. $\Gamma^{-1}(i)$ denotes the set of predecessor nodes of node i , i.e. all j such that $(j,i) \in \mathcal{A}$. Likewise, $\Gamma^{+1}(i)$ denotes the set of successor nodes of node i , i.e. all j such that $(i,j) \in \mathcal{A}$.

Travel times and capacities are assumed to be spatially and temporally independent and independent of one another. During the period of interest $\{0,1,\dots,T\}$, arc attributes may vary with time. After this period, it is assumed that they are stationary, taking the same values as at the last time interval, T . Network \mathcal{N} is permitted to be non-FIFO. It is assumed that if an arc has non-zero probability of zero capacity at time s , then some non-zero probability of zero capacity exists for $t > s$. While a solution when employed in real-time operations may require that some of the supply waits at intermediate nodes, a solution to this problem is not permitted to suggest waiting at these intermediate locations.

Miller-Hooks and Stock Patterson (2004) provide a technique for efficiently converting multi-source, multi-sink network flow problems to single source, single sink problems. Without loss of generality, the SEscape problem is described as a single source, single sink problem for the remainder of this work. The supply at node s at departure time t is denoted by $b_s(t)$ and can take on positive values for any departure time. The demand at node l at departure time t , $b_l(t)$, is zero for all values of t except for $t=T$. No supply is available after time T , i.e. $b_i(t) = 0$, $\forall i \in \mathcal{V}$, $t > T$. It is assumed that all supply can reach the sink no later than time T in all realizations. At departure time T , the demand will equal the total supply B , i.e. $b_l(T) = -B = -$

$\sum_{t=1}^T b_s(t)$. Thus, if flow arrives prior to time T , it simply waits without penalty until time T in order to fulfill the demand. Finally, the supply at all other nodes will be zero at all departure times, $b_i(t) = 0, \forall i \in \mathcal{V} \setminus \{s, l\}, t \in \{0, \dots, T\}$.

The SEscape algorithm determines the pattern of flow that maximizes the minimum path probability of successful arrival at the sink of supply originating at a single source node in dynamic networks with time-varying arc traversal times and STV capacities. The mathematical formulation of the SEscape problem can be written as follows:

$$\text{Max } [\min_{\sigma \in \Omega} \prod_{((i,j),t) \in \sigma} P_{ij}^{x_{ij}(t)}(t)], \quad (1)$$

$$\sum_{j \in \Gamma^{+1}(i)} x_{ij}(t) - \sum_{j \in \Gamma^{-1}(i) \{t | t' + \tau_{ji}(t') = t\}} x_{ij}(t') = b_i(t), \quad \forall i \in \mathcal{V}, t \in \{0, \dots, T\}, \quad (2)$$

$$0 \leq x_{ij}(t) \leq \max_z u_{ij}^z(t), \quad \forall (i, j) \in \mathcal{A}, t \in \{0, \dots, T\}, \quad (3)$$

where Ω is the set of all possible paths from source to sink.

A related formulation is given in Miller-Hooks and Stock Patterson (2004) for the Time-Dependent Quickest Flow Problem (TDQFP) in time-varying but deterministic networks. Constraints (2), i.e. the flow conservation constraints, are identical to those given in Miller-Hooks and Stock Patterson's work. However, because the SEscape problem involves stochastic capacities, the flow to be shipped along each arc $(i, j) \in \mathcal{A}$ at time $t \in \{0, 1, \dots, T\}$ is bounded by the maximum value of all possible capacities on arc (i, j) at departure time t (constraints (3)). Whereas the objective of the TDQFP is to find the flow pattern for completing the shipment with

the minimum total time or minimum evacuation time, the SEscape problem seeks the flow pattern whose minimum success probability path has the maximum value. An example of the mathematical formulation of the SEscape problem as applied on a small network is presented in Appendix B.

4.6 SOLUTION APPROACH

An exact solution methodology for solving the SEscape problem (the SEscape algorithm) is described in this section. The algorithm relies on a probabilistic, time-dependent (PTD) residual network and solution of the Maximum Probability Path (MPP) problem. In this section, details of the PTD residual network and of the SEscape algorithm are given, followed by description of solution to the MPP problem via the MPP algorithm.

4.6.1 The PTD residual network

The SEscape algorithm employs the similar concept of the Time-Dependent Quickest Flow Problem (TDQFP) algorithm (Miller-Hooks and Stock Patterson, 2004) in that it extends the successive shortest path algorithm for solving minimum cost flow problems in static networks (see Ahuja et al, 1993 for additional details on the algorithm) for use in time-varying environments. The basic idea is to iteratively determine the properly defined optimal paths from source to sink in a residual network and incrementally push flow along the paths until all demand is fulfilled.

Unlike the TDQFP algorithm, the SEscape algorithm employs the PTD residual network. For a given flow x , the PTD residual network is denoted $G(x)$. The

arc weight values represent the probability that the capacity of arc (i,j) at time t is not less than n , $P\{u_{ij}(t) \geq n\}$, denoted by $P_{ij}^n(t)$, $n = \{1,2,\dots, \max_z u_{ij}^z(t)\}$. $P_{ij}^n(t)$ can be viewed as the probability of successful arc traversal for n units on arc (i,j) at departure time t . Associated with each arc $(i,j) \in \mathcal{A}$ and departure time $t \in \{0,1,\dots,T\}$ is a residual pointer $\theta_{ij}(t) = x_{ij}(t) - x_{ji}(t + \tau_{ij}(t)) + 1$. This pointer works as an indicator for the remaining capacity, i.e. the remaining capacity of arc (i,j) at time $t = \max_z u_{ij}^z(t) - \theta_{ij}(t) + 1$. Initially, $\theta_{ij}(t)$ is set to 1 for all $(i,j) \in \mathcal{A}$, $t \in \{0,1,\dots,T\}$.

Figure 4.5 demonstrates the construction of the initial PTD residual network.

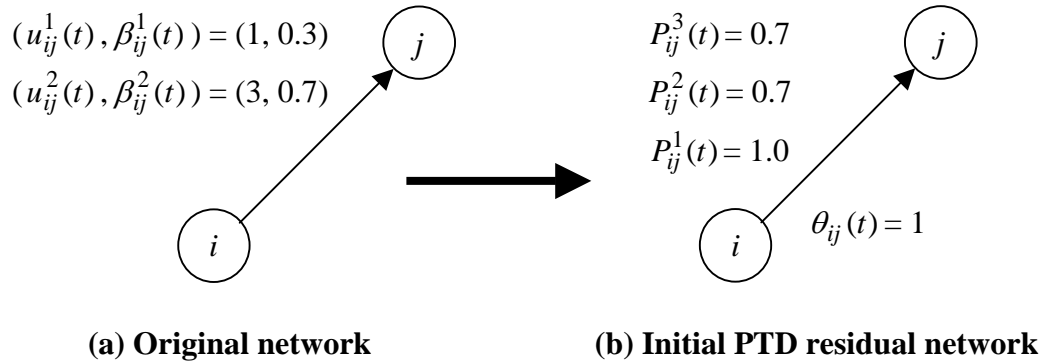


Figure 4.5. The PTD residual network.

In Figure 4.5a, the possible capacities of arc (i,j) at departure time t are 1 with probability 0.3 and 3 with probability 0.7. The PTD residual network transforms this information into the form of $P_{ij}^n(t)$, $n = 1,2,3$. For instance, the probability that two units can successfully traverse the arc at the corresponding departure time is 0.7.

The SEscape algorithm uses backward arcs, whose attributes have a special structure to enable the return of capacity to an arc for canceling decisions made

earlier (ensuring optimality in the larger problem). The backward arcs and the associated probability, $P_{ji}^n(t')$, $t' = t + \tau_{ij}(t) \leq T$, can be implemented as shown in

Figure 4.6.

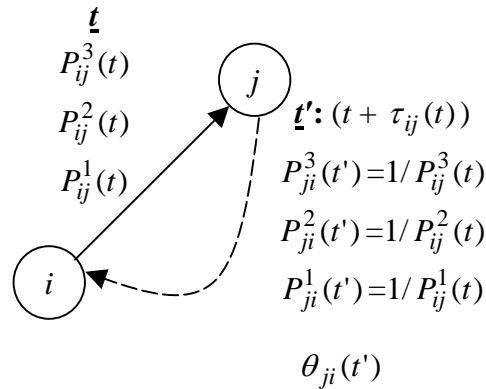


Figure 4.6. The PTD backward arc.

Similar to real arc (i,j) at time t , a residual pointer, $\theta_{ji}(t')$, is associated with backward arc (j,i) at time t' , $t' = t + \tau_{ij}(t) \leq T$, $\forall \{t \in \{0,1,\dots,T\} \mid x_{ij}(t) > 0\}$ to indicate the remaining capacity of the corresponding backward arc at the given time. $\theta_{ji}(t')$ is computed by $x_{ij}(t) - x_{ji}(t')$. Through the special structure of the PTD residual network, including the residual pointers, the probability that each unit will succeed in traversing each arc at a particular time can be readily obtained.

4.6.2 SEscape algorithm

The SEscape algorithm can be decomposed into two main components: 1) generation of paths connecting the source and sink with nonzero probability of having adequate capacity to accommodate all supply units and 2) determination of the number of units to be sent along each path to complete the shipments. In the TDQFP algorithm, at

each iteration, the path with the earliest arrival time in a time-dependent residual network is chosen. In the SEscape algorithm, rather than seek this earliest arrival time path, the algorithm chooses the path with the maximum probability of successfully shipping additional units of flow from the source to sink in the PTD residual network. An overview of the SEscape algorithm is presented next.

1. **Construct a PTD residual network:** Transform the network into a PTD residual network.
2. **Generate Maximum Probability Paths:** Given the PTD residual network, obtain the path with maximum probability for sending one additional unit of flow from the source to the sink by the maximum probability path (MPP) algorithm (described in Subsection 4.5.3). Push one (or more)¹ units of flow along the MPP.
3. **Update the PTD Residual Network:** In the PTD residual network, maintain the values related to $P_{ij}^n(t)$ along each arc (i,j) for each departure time $t \in \{0, \dots, T\}$. Update the residual pointers, $\theta_{ij}(t)$ at each iteration and the backward arcs, where $P_{ji}^n(t')$, $t' = t + \tau_{ij}(t) \leq T$, as needed.
4. **Terminate:** If all supply has been assigned to a path, terminate the algorithm; otherwise, return to step 2.

The SEscape algorithm terminates with a set of paths from the source to the sink and the corresponding number of units to be shipped along each path such that the minimum probability of arrival at the sink is maximized. The actual amount of flow that will be able to pass through each arc at a given point in time is not known until

¹ See how the number of flow units to be shipped on the MPP is determined in Section 4.6.3.

the solution is implemented and values of the arc capacities are revealed. The procedural steps of the SEscape algorithm are provided next. Additional notation employed in the SEscape algorithm is given hereafter.

$e(t)$ = excess supply for departure time t at the source node.

\hat{t} = earliest time t such that supply at the source exists.

Algorithm SEscape

Step 1

Initialize the following variables: $x = 0$, $G(x) = G$ and $e(t) = b_s(t)$, $\forall t \in \{0, \dots, T\}$.

Set $\theta_{ij}(t) = 1$, $\forall (i,j) \in \mathcal{A}$, $t \in \{0, \dots, T\}$.

Step 2

Determine \hat{t} , where $\hat{t} = \min_{t \in \{0, \dots, T\}} (t \mid e(t) > 0)$.

Call the function $MPP(l, T, \theta_{ij}(t), G(x))$, whose output contains ψ_s and σ_s .

Determine ε , where $\varepsilon = \min(e(\hat{t}), \psi_s(\hat{t}))$. If $\varepsilon = 0$, stop, the problem is infeasible.

Augment ε units of flow along path σ_s , i.e. increase $x_{ij}(t)$ by ε , $\forall ((i,j), t) \in \sigma_s$.

Decrease $e(\hat{t})$ by ε .

Step 3

Determine the residual pointers for all $(i,j) \in \mathcal{A}$ given the flow x .

Set $\theta_{ij}(t) = x_{ij}(t) - x_{ji}(t + \tau_{ij}(t)) + 1$, $\forall \{t \in \{0, \dots, T\} \mid t + \tau_{ij}(t) \leq T\}$,²

If $\sum_{t \in \{0, \dots, T\}} e(t) = 0$, **Stop**.

² If there exists real arc $(j,i) \in \mathcal{A}$, the backward arc of arc (i,j) must be differentiated from the original arc (j,i) .

Step 4

Backward arc update:

Add the backward arc (j,i) to the residual graph, $G(x)$, if it does not already exist, for each $(i,j) \in \mathcal{A}$, such that for some $t \in \{0, \dots, T\}$, $x_{ij}(t) > 0$. For all backward arcs, (j,i) , update the following travel times, probabilities of successful arc traversal and residual pointers.

$$\begin{aligned} \tau_{ji}(t') &= -\tau_{ij}(t), \quad t' = t + \tau_{ij}(t) \leq T, \quad \forall \{t \in \{0, \dots, T\} \mid x_{ij}(t) > 0\}, \\ &= T, \quad \forall \{t \in \{0, \dots, T\} \mid x_{ij}(t) = 0\}. \end{aligned}$$

$$P_{ji}^n(t') = 1/P_{ij}^n(t), \quad n = 1, \dots, \max_z u_{ij}^z(t), \quad t' = t + \tau_{ij}(t) \leq T, \quad \forall \{t \in \{0, \dots, T\} \mid x_{ij}(t) > 0\}.$$

$$\theta_{ji}(t') = x_{ij}(t) - x_{ji}(t'), \quad t' = t + \tau_{ij}(t) \leq T, \quad \forall \{t \in \{0, \dots, T\}\}.$$

Return to **Step 2**.

In Step 1, the PTD residual network is constructed from the original network. The excess supply $e(t)$ is initialized to the supply values at each departure time t , and $\theta_{ij}(t)$ is set to 1 for all arcs and departure times. Step 2 features the MPP algorithm for determining the MPP and the appropriate amount of flow to be sent along the path. After the flow has been shipped, the excess supply $e(t)$ is reduced. In Step 3, the residual pointers $\theta_{ij}(t)$ on the constituent arcs along the path are updated: $\theta_{ij}(t) = x_{ij}(t) - x_{ji}(t + \tau_{ij}(t)) + 1$. The final step is concerned with the update of the PTD residual network. Backward arc (j,i) at time $t + \tau_{ij}(t)$ are added in response to the presence of positive flow on arc (i,j) at time t . The travel times, probabilities of successful arc traversal and residual pointers associated with the backward arcs are

updated accordingly. After Step 4, the algorithm returns to Step 2 and this iterative process continues until all the supply has been shipped.

Upon completion, the algorithm provides the set of arc flows, which can be identified through the residual pointers $\theta_{ij}(t)$. That is, the flow on arc (i,j) at departure time t is equal to $\theta_{ij}(t) - 1$. The next subsection describes the MPP algorithm, which is used as a subroutine within the SEscape algorithm.

4.6.3 Maximum Probability Path (MPP) algorithm

The SEscape algorithm relies on the MPP algorithm to determine the MPP and the associated number of units to be shipped along from the source node s to the sink node l in the PTD residual network. The MPP algorithm is a specialized version of the TDLTP algorithm of Ziliaskopoulos and Mahmassani (1993) for determining paths with the maximum probability that one (or more) units can successfully arrive at the sink from each node at each departure time in non-FIFO, time-varying networks. Waiting is not permitted at any node.

The TDLTP algorithm determines the least time paths from all nodes to a desired destination. In the MPP algorithm, in addition to arc travel times, the probability of successful arc traversal is associated with each arc at each departure time and the objective is to determine a path between an origin-destination pair such that the probability of successful arrival at the destination is maximized. Let $P_{ij}(t)$ be the probability that at least one unit can successfully traverse arc (i,j) at departure time t . Assume that such probabilities are independent over space and time. The probability of successful arrival at the destination, given that path a is taken from the

origin, is computed by $\prod_{((i,j),t) \in a} P_{ij}(t)$.

In the MPP algorithm, associated with each node $i \in \mathcal{V}$ and each departure time $t \in \{0,1,\dots,T\}$ are labels $\lambda_i(t)$ and $\psi_i(t)$. Prior to termination, $\lambda_i(t)$ represents a lower bound on the probability of successfully shipping $\psi_i(t)$ units from node i at departure time t to the destination node l . Similar to the original TDLTP algorithm, the MPP algorithm determines the MPPs in an iterative manner by scanning nodes from a scan eligible (SE) list working backward from the destination node. The MPP algorithm constructs the MPP from each node at each departure time through the currently MPP associated with a successor node. If the newly constructed path has greater success probability than the currently MPP from the same node at the same departure time to the destination, this new path will become the MPP.

In each iteration of the SEscape algorithm, the updated PTD residual network $G(x)$ and associated residual capacities $\theta_{ij}(t)$ are used as input to the MPP algorithm. For each arc $(i,j) \in \mathcal{A}$ and each departure time $t \in \{0,\dots,T\}$, $\theta_{ij}(t)$ indicates the appropriate value of the probability of successful arc traversal $P_{ij}^{\theta_{ij}(t)}(t)$ for computing $\lambda_i(t)$. The optimality condition of the MPP algorithm can be written as follows.

$$\lambda_i(t) \geq P_{ij}^{\theta_{ij}(t)}(t) \cdot \lambda_j(t + \tau_{ij}(t)), \text{ for all } j \in \Gamma^+(i) \quad (\text{equation 1})$$

A temporary vector label, $\eta_i(t)$ is employed in the update of $\lambda_i(t)$. If $\eta_i(t) > \lambda_i(t)$, the label is updated, $\lambda_i(t) = \eta_i(t)$, and node i is inserted in the SE list for subsequent scanning. Upon termination, each label $\lambda_i(t)$ provides the maximum

probability that $\psi_i(t)$ units can arrive at the desired destination starting from node i at departure time t . A pointer $\pi_i(t)$ is used to indicate the successor nodes from node i at time t .

The algorithmic steps of the MPP algorithm are given next.

Algorithm MPP ($l, T, \theta_{ij}(t), G(x)$)

Step 1 (Initialization):

Initialize the labels and path pointers.

$$\lambda_i(t) = 0, \forall i \in \mathcal{V} \setminus l, t \in \{0, 1, \dots, T\}.$$

$$\psi_i(t) = \infty, \forall i \in \mathcal{V}, t \in \{0, 1, \dots, T\}.$$

$$\pi_i(t) = \infty, \forall i \in \mathcal{V}, t \in \{0, 1, \dots, T\}.$$

$$D_i(t) = \infty, \forall i \in \mathcal{V}, t \in \{0, 1, \dots, T\}.$$

$$\lambda_l(t) = 1, \forall t \in \{0, 1, \dots, T\}.$$

Create the SE list and insert the destination node l into the SE list.

Step 2 (Select Node for Scanning):

If the SE list is empty, go to step 4. Otherwise, select and delete a node from the SE list. Call this node the current node j .

Step 3 (Update the Node Labels):

For each $i \in \Gamma^{-1}(j)$,

For each $t, \{t \in \{0, 1, \dots, T\} \mid t + \tau_{ij}(t) \leq T, \lambda_i(t) < 1\}$,

$$\eta_i(t) = P_{ij}^{\theta_{ij}(t)}(t) \cdot \lambda_j(t + \tau_{ij}(t)).$$

If $\eta_i(t) > \lambda_i(t)$, then set:

$$\lambda_i(t) = \eta_i(t),$$

$$\pi_i(t) = j \text{ and } SE = SE \cup \{i\},$$

$$\psi_i(t) = \min(\psi_j(t + \tau_{ij}(t)), \kappa),$$

$$\text{where } \kappa = q - \theta, q = \min \{ n \mid n > \theta \text{ and } P_{ij}^n(t) < P_{ij}^{\theta_{ij}(t)}(t) \}.$$

Return to step 2.

Step 4 (Termination):

Upon termination, the MPP algorithm provides the maximum probability λ_i , number of units to be shipped ψ_i and corresponding path σ_i (arcs along the path and the associated departure time from each arc) from all nodes $i \in \mathcal{V}$ to the destination l .

Proofs and the worst-case computational complexity of the MPP algorithm follow directly from those of the original TDLTP algorithm given in Ziliaskopoulos and Mahmassani (1993).

4.6.4 Proof of the SEscape algorithm

In this section, proofs of correctness and computational complexity of the SEscape algorithm are given. For clarity, the proof of Proposition 4.1 is given in the static case. To extend these concepts for networks with time-varying arc traversal times and capacities, the arc travel times, capacities and flow must be expressed with respect to the arrival time at the nodes and the path selection process is performed over the time dimension.

Proposition 4.1. The SEscape algorithm provides the pattern of flow whose minimum path probability of successful arrival of supply at the sink is maximized.

Proof. The two possible cases, one in which no backward arcs are used in the solution and the other in which one or more backward arcs are used in the solution, are considered separately.

Case I: The selected paths employ no backward arc

For shipping n units of flow, the SEscape requires at most n iterations of the MPP algorithm. Because the MPP algorithm guarantees to obtain the maximum probability path, in each iteration the path with the maximum probability of successfully shipping at least one unit from the source to the sink is chosen. Therefore, if the path contains no backward arcs, the success probability of the path for shipping the last set of flow units would have the maximum success probability.

Case II: The selected paths employ one or more backward arcs

When the MPP algorithm selects a path with one or more backward arcs in some iteration, and flow is shipped along the path, capacity is returned to the corresponding arcs, and a new set of paths connecting the source to the sink results. Proof of this second case requires proving that any path in this new set must have a higher probability of successful arrival at the sink than that of any path without backward arcs. An example network given in Figure 4.7 is employed. Suppose n units of flow must be shipped from the source node s to the sink node l and no path employing backward arcs has been introduced into the solution path flows up to $(n-2)$ unit shipments. This implies that the probability of successfully shipping the $(n-2)^{\text{th}}$ unit is the maximum probability of completing the shipment thus far. Let A, B, C, D, E, and

F be the current probabilities of successfully traversing arcs $(s,2)$, $(2,3)$, $(3,l)$, $(s,3)$, $(2,l)$ and (s,l) , respectively.

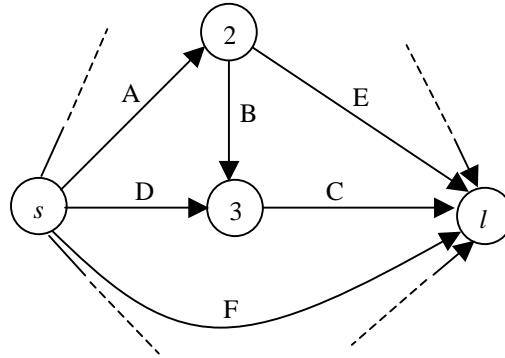


Figure 4.7. The PTD residual network after completion of $(n-2)$ unit shipments.

Assume path $s-2-3-l$ has the maximum probability for shipping the $(n-1)^{\text{th}}$ unit. That is, the following assumptions with respect to the current PTD residual network must hold:

- I) $A \cdot B \cdot C > A \cdot E$. This infers that $B \cdot C \cdot (1/E) > 1$
- II) $A \cdot B \cdot C > D \cdot C$. This infers that $A \cdot B \cdot (1/D) > 1$
- III) $A \cdot B \cdot C > F$

After completing this shipment, backward arcs are introduced in the residual network as portrayed in Figure 4.8.

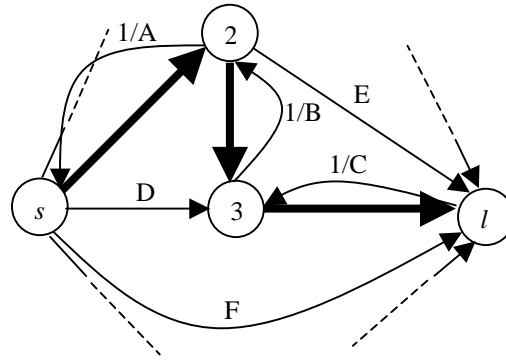


Figure 4.8. The PTD residual network after completion of (n-1) unit shipments.

From Figure 4.8, the updated PTD residual network contains a new path $s-3-2-l$ that employs backward arc $(3,2)$ with corresponding probability $1/B$. For the last shipment, a path with the maximum success probability according to the PTD residual network must be chosen. Two groups of paths are considered here according to whether or not backward arcs are employed. Among all paths that do not contain backward arcs, path $s-l$ is assumed to have the maximum probability (F). We further assume that path $s-3-2-l$ with probability $D \cdot (1/B) \cdot E$ is selected by the MPP algorithm for shipping the last unit, i.e. $D \cdot (1/B) \cdot E > F$. To prove that selecting this path would result in the maximum least probability of successfully arrival at the sink, we must show that the new set of paths created from this path selection, paths $s-3-l$ and $s-2-l$, have higher probabilities of successfully completing this last shipment than that of path $s-l$. In other words, we need to prove that $C \cdot D > F$ and $A \cdot E > F$. Since path $s-3-2-l$ is selected over path $s-l$, the following condition holds.

$$D \cdot (1/B) \cdot E > F \quad (1)$$

Equation (1), $C \cdot D \cdot (1/B) \cdot E > C \cdot F$, infers that

$$C \cdot D > [B \cdot C \cdot (1/E)] \cdot F \quad (2).$$

Given assumption I,

$$B \cdot C \cdot (1/E) > 1 \quad (3).$$

Through (2) and (3), we can conclude that

$$C \cdot D > F \quad (4).$$

A similar process can be applied for proving $A \cdot E > F$.

Equation (1), $A \cdot D \cdot E > A \cdot B \cdot F$, infers that

$$A \cdot E > [(A \cdot B \cdot (1/D))] \cdot F \quad (5).$$

Given assumption II,

$$A \cdot B \cdot (1/D) > 1 \quad (6).$$

Through (5) and (6), we can conclude that

$$A \cdot E > F \quad (7).$$

If a path employing backward arcs is selected by the MPP algorithm, that path will result in the maximum probability of successful arrival at the sink. Therefore, the SEscape algorithm guarantees to provide the pattern of flow whose minimum path probability of successful arrival of supply at the sink is maximized. ♦

Proposition 4.2. The worst-case computational complexity of the SEscape algorithm is $\sim O(B \cdot F)$, where B is the number of supply units, and F is the running time of the embedded path finding algorithm.

Proof. In the worst case, the SEscape algorithm requires B runs of the path finding algorithm for B supply units. Because the complexity of the path finding algorithm is

$\sim O(F)$, the worst-case computational complexity of the SEscape algorithm is $\sim O(B \cdot F)$.

The MPP algorithm embedded in the SEscape algorithm for finding paths has the similar complexity to the TDLTP algorithm, i.e. $\sim O(V^3 \cdot I^2)$, where $V = |\mathcal{V}|$ is the number of nodes in the network and I is the number of time intervals considered. Therefore, the worst-case computational complexity of the SEscape algorithm is $\sim O(B \cdot V^3 \cdot I^2)$. ♦

It is significant to note that while each iteration of the SEscape algorithm requires only a single path for shipping flow from the source to the sink at a particular time, the MPP algorithm finds the MPP from all nodes to a destination for all departure times. The complexity of the SEscape algorithm can be considerably improved if a similar algorithm were to be discovered that can avoid the steps required in determining the unnecessary solutions, i.e. solutions from nodes or at departure times that are not needed.

4.7 ILLUSTRATIVE EXAMPLE

In this subsection, the procedural steps of the SEscape algorithm are illustrated on a simple example network displayed in Figure 4.9. Three units of flow must be shipped from node 1 at time $t=0$ to node 4, i.e. $b_1(0) = 3$. The corresponding arc travel times and capacities that are necessary for determining the optimal pattern of flow for completing the shipment are provided in Table 4.1. Table 4.2 shows the associated $P_{ij}^n(t)$ values for the initial PTD residual network. For instance, the probability that

the capacity of arc (1,2) at time $t=0$ is not less than three units is 0.3, i.e. $P_{12}^3(0) = 0.3$.

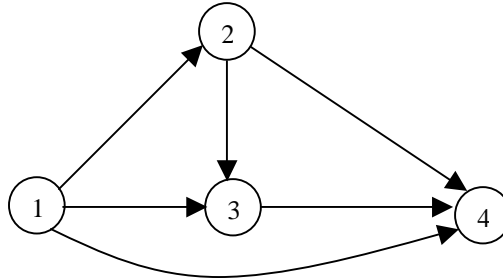


Figure 4.9. Example network.

Table 4.1. Arc travel times and capacities for the example network.

(i,j)	(1,2)	(1,3)	(1,4)	(2,3)	(2,4)	(3,4)
$\tau_{ij}(t)$	$t=0:$ 2	$t=0:$ 4	$t=0:$ 6	$t=2:$ 2	$t=2:$ 3	$t=4:$ 2
$u_{ij}^z(t) (\beta_{ij}^z(t))$	$t=0:$ 4(0.3) 2(0.5) 1(0.2)	$t=0:$ 1(0.4) 0(0.6)	$t=0:$ 3(0.1) 0(0.9)	$t=2:$ 2(0.9) 1(0.1)	$t=2:$ 2(0.1) 1(0.2) 0(0.7)	$t \geq 4:$ 2(0.90) 1(0.05) 0(0.05)

Table 4.2. The PTD residual network.

(i,j)	(1,2)	(1,3)	(1,4)	(2,3)	(2,4)	(3,4)	
$P_{ij}^n(t)$	n	$t=0$	$t=0$	$t=0$	$t=2$	$t=2$	$t \geq 4$
	5	0	0	0	0	0	0
	4	0.3	0	0	0	0	0
	3	0.3	0	0.1	0	0	0
	2	0.8	0	0.1	0.9	0.1	0.90
	1	1	0.4	0.1	1	0.3	0.95
	0	1	1	1	1	1	1

The following details the steps of the SEscape algorithm on this example problem for shipping three units from node 1 to node 4, departing node 1 at time $t=0$.

Iteration 1**Step 1**

Initialize the following variables: $x=0$, $G(x) = G$ and $e(t) = b_1(t)$, $\forall t \in \{0, \dots, 6\}$.

Set $\theta_{ij}(t) = 1$, $\forall (i,j) \in \mathcal{A}$, $t \in \{0, \dots, 6\}$.

Step 2

$\hat{t} = 0$.

Call the function $MPP(l = 4, T = 6, \theta_{ij}(t), G(0))$,

$\lambda_1(0) = 0.95$, $\psi_1(0) = 1$ and $\sigma_1(0) = ((1,2), 0) ((2,3), 2) ((3,4), 4)$.

$\varepsilon = \min(e(0), \psi_1(0)) = 1$.

Augment one unit of flow along path $\sigma_1(0)$, $x_{12}(0) = x_{23}(2) = x_{34}(4) = 1$.

Decrease $e(0) = 3 - \varepsilon = 2$.

Step 3

Determine the residual pointers for all $(i,j) \in \mathcal{A}$ given the flow x :

$\theta_{12}(0) = \theta_{23}(2) = \theta_{34}(4) = 2$.

Step 4

Add backward arcs (2,1) (3,2) and (4,3) to the residual graph:

Table 4.3. Backward arc information.

(i,j)	n	(2,1)	(3,2)	(4,3)
		$t=2$	$t=4$	$t=6$
$P_{ij}^n(t)$	5	0	0	0
	4	1/0.3	0	0
	3	1/0.3	0	0
	2	1/0.8	1/0.9	1/0.90
	1	1	1	1/0.95
	0	1	1	1

$$\tau_{21}(2) = -2, \quad \tau_{32}(4) = -2, \quad \tau_{43}(6) = -2.$$

$$\theta_{21}(2) = \theta_{32}(4) = \theta_{43}(6) = 1.$$

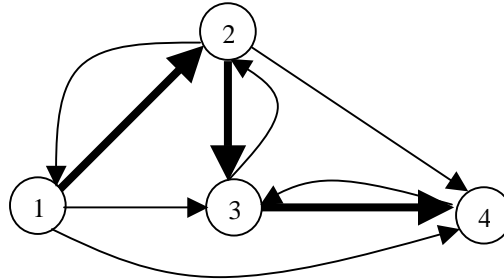


Figure 4.10. PTD residual network.

Iteration 2

Step 2

Call the function $MPP(l = 4, T = 6, \theta_{ij}(t), G(1))$,

$$\lambda_1(0) = 0.648, \quad \psi_1(0) = 1 \quad \text{and} \quad \sigma_1(0) = ((1,2), 0) ((2,3), 2) ((3,4), 4).$$

$$\varepsilon = \min (e(0), \psi_1(0)) = 1.$$

Augment one unit of flow along path $\sigma_1(0)$, $x_{12}(0) = x_{23}(2) = x_{34}(4) = 2$.

Decrease $e(0) = 2 - \varepsilon = 1$.

Step 3

Determine the residual pointers for all $(i,j) \in \mathcal{A}$ given the flow x :

$$\theta_{12}(0) = \theta_{23}(2) = \theta_{34}(4) = 3.$$

Step 4

$$\theta_{21}(2) = \theta_{32}(4) = \theta_{43}(6) = 2.$$

Iteration 3

Step 2

Call the function $MPP(l = 4, T = 6, \theta_{ij}(t), G(2))$,

$$\lambda_1(0) = 0.133, \psi_1(0) = 1 \text{ and } \sigma_1(0) = ((1,3), 0) ((3,2), 4) ((2,4), 2).$$

$$\varepsilon = \min (e(0), \psi_1(0)) = 1.$$

Augment one unit of flow along path $\sigma_1(0)$, $x_{13}(0) = x_{32}(4) = x_{24}(2) = 1$.

Decrease $e(0) = 1 - \varepsilon = 0$.

Step 3

Determine the residual pointers for all $(i,j) \in \mathcal{A}$ given the flow x :

$$\theta_{12}(0) = \theta_{34}(4) = 3, \theta_{23}(2) = \theta_{13}(0) = \theta_{24}(2) = 2.$$

Since $\sum_{t \in \{0, \dots, 6\}} e(t) = 0$, stop.

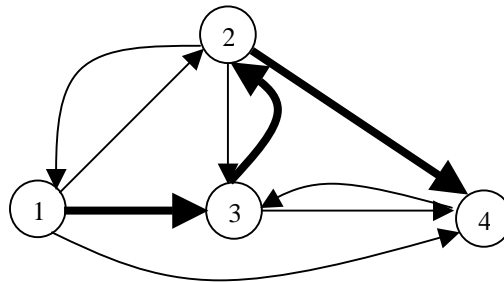


Figure 4.11. Shipping one unit on path 1-3-2-4.

The optimal set of path flows for completing the shipment of three units from node 1 to node 4 is to split the flow, one unit along each path: 1-2-4, 1-2-3-4 and 1-3-4, as depicted in Figure 4.12. This pattern of flow guarantees that the minimum

success probability has the maximum value. The minimum probability of successful arrival at node 4 by following this solution is 0.24 through path 1-2-4.

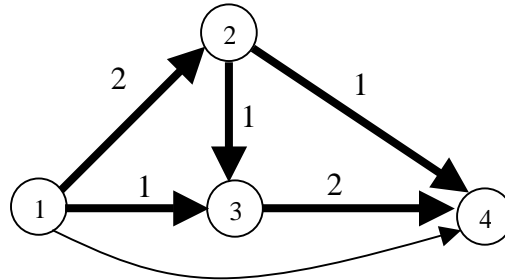


Figure 4.12. Optimal solution.

Section 4.8 presents computational results for assessing the computational performance of the SEscape algorithm.

4.8 NUMERICAL EXPERIMENTS

In this section, computational experiments are conducted through randomly generated networks to examine the average run time and other characteristics of the SEscape algorithm. The algorithm was coded in C++ and run on a DEC Alpha XP1000 professional workstation with 1 gigabyte ram and 2 gigabyte swap, running Digital 4.0E operating system, using Digital's C++ compiler.

4.8.1 Experimental Design

The same network configurations as provided in Miller-Hooks and Stock Patterson (2004) for testing the TDQFP algorithm were used. That is, the networks have either 25, 100 or 500 nodes. The average in- and out- degree for each node is approximately

4, varying from 2 to 9. For each network topology, three sets of the number of time intervals in the non-stationary period are considered: 60, 120 and 240. Supply exists at one-quarter of the time intervals in the period of interest. Five supply units were assigned to each selected interval.

The arc travel times for each time interval are randomly generated between 1 and 15 time units and are integral. Three random arc capacities for each time interval range from 1 to 20 units and also are integral. For each network configuration, a single source was randomly chosen and 10 runs were completed, corresponding to 10 randomly selected sinks. The average of these 10 runs was recorded. This requires 90 runs of 9 different network configurations.

4.8.2 Experimental Results

The results of all runs are reported in Table 4.4. In the table, the first two columns for each combination of network topology and the number of time intervals demonstrate the run times of the SEscape and MPP algorithms, respectively. The run times do not include I/O time, as is common practice in reporting such average run times. In the third column, the number of paths required in completing the shipments is presented. The average values of the 10 runs are given in bold at the end of each column. Note that for the 100 node network with 60 time intervals in the non-stationary period, the time bound T had to be extended to 75 to accommodate all shipments. Similarly, T was extended to 90 and 150 for the 500 node network with 60 and 120 time intervals, respectively.

Table 4.4. Experimental results.

25								
60			120			240		
SEscape	MPP	# of paths	SEscape	MPP	# of paths	SEscape	MPP	# of paths
0.933	0.917	32	4.866	4.783	66	30.949	30.649	179
0.900	0.883	32	7.150	7.100	92	35.032	34.799	192
0.850	0.833	31	5.500	5.450	81	24.449	24.349	150
1.100	1.067	40	5.883	5.833	86	30.599	30.415	165
1.433	1.367	44	4.650	4.600	65	35.049	34.799	194
1.267	1.233	46	4.666	4.600	78	23.766	23.666	145
0.967	0.967	37	7.166	7.116	96	23.449	23.299	156
0.983	0.950	38	6.333	6.283	90	17.199	17.016	116
1.567	1.550	43	7.083	6.983	99	21.766	21.516	147
0.983	0.967	37	5.566	5.500	85	30.016	29.799	173
1.098	1.073	38	5.886	5.825	83.8	27.227	27.031	162
100								
60(75)			120			240		
SEscape	MPP	# of paths	SEscape	MPP	# of paths	SEscape	MPP	# of paths
6.550	6.516	32	41.415	41.165	104	201.342	200.592	233
7.000	6.950	34	37.549	37.382	99	158.127	157.460	206
8.416	8.333	40	37.282	37.032	104	167.943	167.293	213
10.566	10.550	49	40.632	40.432	110	141.461	140.794	197
6.483	6.450	30	38.365	38.182	105	145.694	145.011	196
11.016	10.983	50	31.182	31.049	91	134.245	133.695	177
9.516	9.450	48	40.382	40.182	108	133.828	133.178	186
9.600	9.516	49	35.732	35.565	99	115.612	114.995	163
9.600	9.450	49	38.682	38.448	102	151.044	150.294	206
10.733	10.600	52	45.782	45.481	111	138.161	137.645	187
8.948	8.880	43.3	38.700	38.492	103	148.746	148.096	196
500								
60(90)			120(150)			240		
SEscape	MPP	# of paths	SEscape	MPP	# of paths	SEscape	MPP	# of paths
78.264	77.964	50	330.437	329.421	118	1122.390	1118.721	239
84.363	84.030	55	413.183	411.934	130	1194.690	1190.770	248
97.229	96.830	55	306.738	305.555	115	1242.780	1238.984	251
79.397	79.030	51	343.220	342.203	122	1104.740	1101.172	231
83.497	83.213	54	332.587	331.520	117	1895.420	1891.325	254

78.414	78.047	49	338.270	337.087	118	1337.900	1333.781	253
96.846	96.529	60	354.536	353.186	123	1054.040	1050.575	226
79.947	79.697	51	324.004	322.937	120	1247.050	1243.101	254
78.814	78.514	52	308.704	307.671	113	1002.890	999.177	227
80.097	79.680	51	299.888	298.938	107	1277.920	1274.067	254
83.687	83.353	52.8	335.157	334.045	118	1247.982	1244.167	244

The results in Table 4.4 show that significant portion of the computational time is due to calls to the MPP algorithm as the average run time of the MPP algorithm takes over 99 percent of the total run time required by the SEscape algorithm. An algorithm with less overhead (i.e. that seeks the path from one source to one sink at one departure time) could reduce this ratio. Given the same amount of supply, the average number of paths required in shipping all the supply increases with the size of network. The SEscape algorithm performs well even on networks with 500 nodes and much better than predicted by the worst-case complexity, $\sim O(B \cdot V^3 \cdot I^2)$, given in Proposition 4.2. For instance, the average run time for the 500 node network with 240 time intervals is 1,244 c.p.u. seconds, which is less than 8,000 times the average run time of the 25 node network with the same number of time intervals (27 c.p.u. seconds), that would be expected based on worst-case complexity.

The SEscape algorithm requires significantly more computational effort than does the TDQFP algorithm for problems of similar size. For instance, the average run time of the SEscape algorithm on the network with 500 nodes and 240 time intervals is 1,244 c.p.u. seconds; whereas the TDQFP algorithm requires 19 c.p.u. second as reported in Miller-Hooks and Stock Patterson (2004). Many factors contribute to this increase in required computational effort of the SEscape algorithm over that of the TDQFP algorithm. First, the number of paths required in the SEscape problem for

sending all the supply is frequently larger than that required in the TDQFP algorithm. This is because the TDQFP algorithm always pushes flow up to the maximum capacity of the chosen path, and thus, will likely require fewer paths for completing all shipments. The SEscape algorithm, on the other hand, assigns flow along the path for the amount that maximizes the probability of successful arrival at the sink. Second, with the same amount of supply, larger T bounds are required to solve the SEscape problem, which directly affects the computational steps of the MPP algorithm. Lastly, the computational complexity of the MPP algorithm, $\sim O(V^3 \cdot I^2)$, is worse than that of the TDEAT algorithm, $\sim O(V^2 \cdot I)$, the path algorithm used within the TDQFP algorithm.

Note that the TDEAT algorithm allows waiting and cannot be used with the SEscape algorithm. However, the difference in the average run time between the SEscape and the TDQFP algorithms is not as large as predicted by worst-case complexity analyses. For example, the average run time of the SEscape algorithm for the 500 node network with 240 time intervals requires 1,244 c.p.u. seconds, which is less than 120,000 times the average run time of the TDQFP algorithm for the same network configuration (19 c.p.u. seconds), as would be expected by its worst-case computational complexity.

4.9 CONCLUSIONS

In this chapter, the SEscape problem is formulated and an exact solution approach is proposed. The SEscape problem is concerned with the determination of the optimal evacuation instructions whose minimum probability of successful arrival of the

evacuees at the safe location is maximized. The SEscape algorithm explicitly considers the time-varying and uncertain nature inherent in such situations. Specifically, it determines the set of *a priori* path flows in capacitated dynamic networks, where arc traversal times are time-varying, and arc capacities are discrete random variables whose probability distribution functions vary with time. Following the instructions given by the SEscape algorithm would maximize the likelihood that any person who is subject to the greatest risk will succeed in escaping. Results of extensive numerical tests show that the SEscape algorithm performs significantly better than expected by its worst-case computational complexity. The proposed conceptual framework and algorithmic steps can be used in evacuation planning, enabling safer evacuation of a building in the event of military attack, fire, natural disaster, or other circumstances warranting quick escape.

CHAPTER 5

HEURISTICS FOR MSTV CAPACITATED NETWORKS

5.1 INTRODUCTION

The SEscape algorithm was proposed in Chapter 4 for determining the pattern of flow that maximizes the minimum path probability of successful arrival of supply at the sink. Development of the SEscape algorithm was motivated by the need to provide instructions to evacuees in the event that rapid evacuation of a large damaged or burning building is required. While the algorithm addresses stochastic, time-varying (STV) capacities, the time-varying arc travel times are assumed to be known deterministically. In this chapter, network flow problems in dynamic networks, where knowledge of arc travel times is uncertain, are addressed.

Explicit consideration of stochastic and time-varying travel times makes the SEscape problem and other related problems (e.g. the problem of determining evacuation routes with the minimum total travel time) significantly more difficult. This is because arc travel times are known at best only probabilistically and, therefore, the location of shipped flow units at any point in time cannot be identified with certainty. It appears that no existing works in the literature address such problems without relying on simulation. Thus, an exact solution (given discrete random arc weights) would require enumeration of every potential combination of arc travel times and capacity realizations. Such an approach would require enormous computational effort. In this section, a technique that can provide competitive approximate solutions with significantly less computational effort is proposed. A

genetic algorithm (GA) is presented for determining optimal path flows with respect to several problems in dynamic networks, where arc traversal times and capacities are random variables with probability mass functions that vary with time. Capacitated networks with such stochastic and time-varying arc travel times and capacities are referred to herein as STV capacitated networks.

A GA, like any meta-heuristic, cannot guarantee an optimal solution. To assess the performance of the developed GA, one must compare solutions generated by that technique to exact solutions or, a bound on the exact solution. However, no efficient technique has been proposed to determine the exact solution or bounds on the solution for network flow problems in STV capacitated networks. In this chapter, the framework for the GA is first presented for solving the problem of determining optimal path flows, where the arc travel times are assumed to be deterministic and time-dependent. The solution approach can be used to seek a pattern of flow for shipping a given amount of supply such that a single objective is achieved, e.g. minimum total time, maximum expected flow or maximum minimum path probability of successful arrival at the sink (the SEscape problem). Numerical experiments were conducted to assess the performance of the proposed GA. Results of the experiments show that the GA results in solutions of high quality as compared with exact solutions generated by exact approaches. The GA is then extended for use in more complex STV and MSTV, capacitated networks, where no exact algorithm exists. An MSTV capacitated network is an STV capacitated network with multiple STV arc attributes.

GAs are powerful stochastic search techniques that rely on the concept of natural selection and evolution. One of the advantages of GAs over traditional optimization techniques is that optimal solutions are sought from the entire decision space. Therefore, GAs have been used in many published works to address a variety of complicated combinatorial optimization problems, such as non-linear, discrete, stochastic, or multiobjective problems, which cannot be efficiently solved by currently existing exact techniques.

Each solution in a GA is represented by a chromosome. Each chromosome is characterized by a series of genes (i.e. decision variables). A GA generally starts by randomly generating an initial set of solutions called a population. New chromosomes are produced through successive populations called generations. In each generation, a crossover operator recombines two chromosomes to form new chromosomes called offspring. The offspring may be perturbed using a mutation operator that randomly changes one or more elements in a chromosome. Potentially good solutions with respect to a fitness function will be selected to contribute to the next generation and further reproduction. The entire process is repeated until the termination criteria are met (see, for example, Goldberg (1989) for additional detail on GAs).

The greatest difficulty in applying GAs to network flow problems lies in appropriate design of solution representation and constraint handling. A well designed representation enables the underlying genetic operators to perform efficiently in exploring better solutions. Binary strings of 1s and 0s are widely used to represent decision variables in GA applications, because of the flexibility in encoding and recombining solutions. That is, integer and real valued variables are transformed

into binary strings and new solutions are created by exchanging fragments of two strings (during crossover) between breaking points.

One of many problems encountered in applying GAs to a network flow problem is constraint handling. Typical constraints of network flow problems include flow conservation constraints (i.e. inflow is equal to outflow at every transshipment node) and capacity constraints (i.e. the amount of flow shipped along an arc cannot exceed the capacity of that arc). Any solution that violates any of the constraints is infeasible. Several techniques have been proposed in the literature for handling constraints, for example, penalty functions, repair algorithm and constraint preserving operators. Appropriate selection of a constraint handling method is a crucial step in development of a GA.

While GAs have been used in a broad range of application areas, few works apply GAs to network flow problems, perhaps due to the availability of exact and efficient optimization techniques. Vignaux and Michalewicz (1991) developed a GA to solve the transportation problem. In the transportation problem, each node is either a source or sink, i.e. no transshipment nodes exist. The objective is to find the amount to be shipped from the source nodes to the sink nodes such that the total cost is minimized. A constraint is associated with each source to specify the available supply and each sink to ensure the demand is satisfied. Capacity constraints are omitted in this work. Two representation schemes were proposed: vector and matrix based structures. While these representation structures are suitable in this context, they cannot be directly extended for use in addressing problems that involve transshipment nodes. The genetic operators are designed such that only feasible

solutions are generated in each population. Such operators will fail to maintain feasibility if capacity constraints are imposed.

Gen et al. (2001) summarized several works on network design problems that have been addressed by GAs. These problems include: the fixed charge transportation problem, degree-constrained minimum spanning tree problem, and shortest path problem. In the first problem, the network structure is represented by a Prufer number. A checking step is embedded in the procedure to prevent the generation of infeasible solutions during the intermediate steps of the procedure. The second problem deals with the minimum spanning tree problem with side constraints that limit the degree of each node. A two-dimensional structure was proposed to encode a spanning tree: one dimension for node permutations and the other for degree constraints. The constraints are handled by setting up conditions to keep only feasible solutions in every step of the process. In the last problem, a GA for solving the bi-criteria shortest path problem in static networks is discussed. The difficulties of encoding a path are twofold: 1) a solution may contain a large number of repeated nodes, resulting in cycles; and 2) a random series of arcs might not form a path. A priority-based encoding method was proposed to resolve these difficulties. The position of each gene represents a node and the corresponding value of the gene represents the priority of that node in the path. The disadvantage of this encoding scheme is that different chromosomes may result in the same path, which in turn decreases the diversity of solutions.

Davies and Lingras (2003) developed a GA for solving shortest path problems with and without waiting at nodes in a real-time environment (i.e. where updated

travel times are received). The algorithm is composed of two components: Prediction Module and GAs for rerouting shortest paths. The Prediction Module component provides the updated travel times. The GA component determines the updated shortest path given the new information. In the GA, each chromosome represents a path between the origin and destination and each gene represents a node in the path. Such a technique guarantees that every intermediate solution is feasible.

All of the works discussed thus far consider only uncapacitated networks. Few works have proposed GAs for network flow problems in capacitated networks. Munakata and Hashier (1993) addressed the maximum flow problem using a GA. Each solution is represented by a flow matrix. The algorithm starts by randomly generating the initial feasible population. Two criteria, flow balance at nodes and saturation rate of the flow (i.e. the proportion of the flow to the maximum arc capacity), are incorporated in the fitness function. To generate a new solution, two solutions are randomly selected from the population according to the probability computed by the ratio of the solution fitness to the sum of the fitness of the population. For each pair of solutions, the crossover operator compares each node in one solution to the same node in the other and the arcs associated with the node that is better in the two criteria will be selected to form a new solution. If none is best with respect to both criteria, random selection is performed. The mutation operator perturbs a solution by randomly adjusting the flow of each arc in increments of 1. Unlike the aforementioned works, constraints are not addressed through these two operators, inevitably affecting the feasibility of flow patterns. Infeasible intermediate solutions are penalized by decreasing their fitness value. Experimental results show

that the GA is inefficient in comparison with already available exact procedures. The number of generations required to determine optimal or near-optimal solutions is large. Moreover, the algorithm frequently converges to an infeasible solution.

Sadek et al. (1997) developed a GA for addressing dynamic traffic assignment, where traffic flow limitations imposed by capacity constraints are explicitly considered. The objective function is to minimize the total time that vehicles spend en route to their destinations. A chromosome is represented by a real-valued vector, each element of which is the number of vehicles that are assigned to an arc during a time period. The constraints are classified into two groups: those that are met in the generation of the initial solutions and those that are not met, but that are addressed by a penalty function method. The authors found that the GA required less than a third of the time required by Microsoft Excel Solver to obtain similar results in experiments on a small (nine-arc, six-node) network.

Most of the GAs discussed thus far ignore capacity constraints. Some works generate initial feasible solutions, but are not guaranteed to produce a feasible solution at the end. None of these works has considered the time-varying and uncertain nature of the network attributes. Note too, as discussed in Chapter 4, that no exact algorithm has been proposed to address network flow problems in STV capacitated networks. Only a few works have employed simulation models to address such problems.

The aim of this work is to develop a GA framework for determining *a priori* path flow decisions to ship supply from a source to a sink such that the total cost is minimized. The problem is considered in a dynamic, STV capacitated network. In

Section 5.2, the minimum cost dynamic flow problem with deterministic, time-varying arc attributes (i.e. travel times and capacities) is addressed. A noisy GA is presented in Section 5.3 that extends the proposed GA for solving the minimum cost flow problem, where arc travel times and capacities are random variables whose probability distribution functions vary with time. In Section 5.4, the GA is further extended for use in MSTV capacitated networks.

5.2 A GENETIC ALGORITHM FOR DETERMINISTIC, TIME-VARYING NETWORKS

In this section, a GA is developed for solving the minimum cost dynamic flow problem in deterministic, time-varying networks. No waiting is permitted at any node. Without loss of generality, the proposed GA can be used with many other objectives, e.g. the SEscape problem, where arc capacities are known only probabilistically. In many of the works that have proposed GAs for solving deterministic and static network flow problems, the performance of the GA was not particularly impressive. That is, the optimal solutions are not guaranteed and if they are found, tremendous computational effort is required. The purpose of this section is not to develop a GA to compete with conventional algorithms, but to design a framework that can be extended to solve a more complicated problem, where no exact method is available, e.g. the minimum cost dynamic flow problem in STV capacitated networks.

5.2.1 Network Notation and Problem Definition

Let $\mathcal{G} = (\mathcal{V}, \mathcal{A}, \{0,1,\dots,T\})$ be a finite digraph, where \mathcal{V} is the set of nodes, \mathcal{A} is the set of directed arcs connecting the nodes and $\{0,1,\dots,T\}$ is the time frame of interest discretized into small time intervals. For each arc $i \in \mathcal{A}$ and departure time $t \in \{0,1,\dots,T\}$, $u_i(t)$ and $\tau_i(t)$ denote the associated capacity and traversal time, respectively. A single source and a single sink are denoted by node s and node l , respectively. $b_s(t)$ represents the supply at the source node s at time t .

The problem addressed in this section seeks the paths along which to send a given supply from a single source to a single sink such that a single objective is achieved. Two network flow problems are considered. The first problem seeks the pattern of flow that minimizes the total time spent completing all shipments in time-dependent, dynamic networks, i.e. a variation on the time-dependent quickest flow problem (TDQFP) of Miller-Hooks and Stock Patterson (2004). The second problem is to maximize the minimum path probability that supply will successfully arrive at the sink, i.e. the SEscape problem. The proposed GA framework can be employed with many other objectives and will be extended for use in STV and MSTV capacitated networks in Sections 5.3 and 5.4, respectively.

5.2.2 Genetic Algorithm

There are six main components to be designed in developing a GA: 1) solution representation, 2) initialization, 3) crossover, 4) mutation, 5) evaluation/selection, and 6) criteria for termination. The details of each component embedded in the proposed GA are provided hereafter.

Representation of Solutions

Several techniques can be used for constraint handling, including penalty functions, repair algorithm and constraint preserving operators. In this work, the constraints are addressed through the encoding scheme. The solution representation plays an important role in accommodating the proposed encoding scheme.

There are a variety of ways to represent a chromosome. Rather than relying on the commonly used bit strings, a more sophisticated structure is developed. The representation structure is specifically designed to encode only solutions that abide by the enforced constraints. Each solution contains several paths together with corresponding flow, forming a flow pattern for shipping certain supply. Each gene consists of two parts. The first part contains a sequence of arcs forming a path from the source to the sink. The second part indicates the number of flow units to be sent through the path. The representation of a solution is indicated by the set $\{ P_i(t), F_i(t) \}$, for $i = 1, 2, \dots, p$, where p is the total number of paths required to complete the shipments. Note that the number of paths required to complete the shipments depends on the level of flow shipped along each chosen path.

$P_i(t) = (a_i^1(t), a_i^2(t), \dots, a_i^n(t))$ denotes a path consisting of n arcs from source node s to sink node l , where $a_i^j(t)$ represents the j^{th} arc in path i departing from the source at time t . $a_i^{j+1}(t)$ is a successor arc of $a_i^j(t)$, and $a_i^1(t)$ is the first arc on the path. $F_i(t)$ denotes the associated number of flow units to be sent along $P_i(t)$.

Figure 5.1 illustrates the representation of a chromosome for a flow pattern, where three flow units are to be shipped from node s to node l , departing from the origin at time $t = 0$ in a network consisting of four nodes and five arcs. Assume two units are shipped on path 1-2 and one unit is shipped on path 1-4-5 as shown in the figure.

	Gene 1	Gene 2
Chromosome	$P_1(0), F_1(0)$	$P_2(0), F_2(0)$

Gene 1: $\{ P_1(0), F_1(0) \} = \{(1,2), 2\}$

Gene 2: $\{ P_2(0), F_2(0) \} = \{(1,4,5), 1\}$

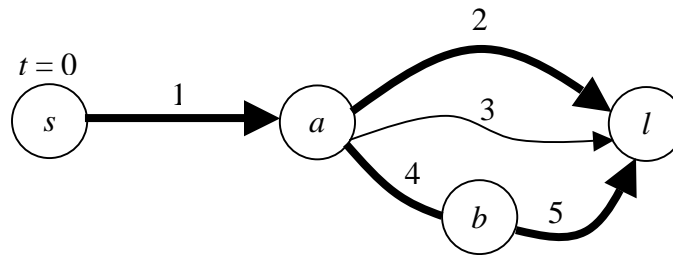


Figure 5.1. Solution representation.

Given this representation structure, different solutions may have a different number of genes depending on the associated paths and flow levels. In addition to assisting in constraint handling, another advantage of this representation scheme is that only portions of the network, on which the flow is assigned, are presented in a chromosome. For example, in Figure 5.1, the arc with no flow, e.g. arc 3, is not

included in the chromosome. This avoids consideration of insignificant portions of the solution in subsequent recombination.

Initialization

The encoding scheme employed in the initialization operator is designed to generate the initial population. For each supply time t , a set of paths is randomly selected for shipping all supply units from the source to the sink. The detailed steps of the initialization can be stated as follows.

For each supply time t , $\{t \in S \mid b_s(t) \neq 0\}$,

1) Set $i=1$.

2) Randomly select a path $P_i(t) = (a_i^1(t), a_i^2(t), \dots, a_i^n(t))$ from the source to the sink such that the associated amount of flow $F_i(t) \neq 0$. $F_i(t)$ can be identified using the following equation.

$$F_i(t) = \min (b_s(t), u_i^1(t), u_i^2(t), \dots, u_i^n(t)) \quad (1),$$

where $u_i^j(t) =$ corresponding capacity of $a_i^j(t)$.

(i.e. $u_i^j(t) = u_k(t_j)$, $k = a_i^j(t)$, $t_1 = t$, $t_{j+1} = t_j + \tau_k(t_j)$, $j = 1, 2, \dots, n$).

3) Update the remaining capacity along each arc: $u_k(t_j) = u_k(t_j) - F_i(t)$,

where $k = a_i^j(t)$, $t_1 = t$, $t_{j+1} = t_j + \tau_k(t_j)$, $j = 1, 2, \dots, n$.

4) Update remaining supply to be shipped at the source: $b_s(t) = b_s(t) - F_i(t)$. If $b_s(t) \neq 0$, set $i = i + 1$ and return to Step 2.

With this initialization technique, capacity constraints and flow conservation at transshipment nodes are satisfied. The flow conservation at the source and sink are

fulfilled by repeating the entire process until all supply units are shipped. In the case that the path flows selected thus far cannot accommodate the remaining supply, i.e. no path with non-zero capacity exists between the source and the sink after the last shipment, the last path introduced into the chromosome is removed and replaced with new paths by repeating Step 2 until all the demand is fulfilled. This initialization process guarantees that only feasible solutions are generated in the first population.

Crossover

Offspring are produced from parents selected from the current population in crossover (mating). It is possible that the newly generated solutions (i.e. offspring) will violate the problem constraints if crossover is not designed to maintain feasibility. Therefore, the crossover operator proposed here is developed such that only feasible solutions are generated.

In this work, the crossover operator combines two randomly selected solutions to form a single child. For each pair, the following steps are performed. For each supply time t , $\{t \in S \mid b_s(t) \neq 0\}$,

- 1) Determine the ratio of path cost to path capacity for each path in the chosen parents.
- 2) Rank all paths according to the ratio of path cost to path capacity in non-decreasing order.
- 3) Pick the first path from the rank and determine the flow units to be shipped on that path by Equation 1.
- 4) Eliminate the chosen path from the rank. Update the remaining capacity of all arcs in that path.

5) Update $b_s(t)$. If $b_s(t) \neq 0$, return to Step 3.

The method of ranking paths according to their path cost to path capacity ratios gives higher priority to paths with lower cost per flow unit for inclusion in the new solution. This feature enables the algorithm to explore promising regions of solution space and can be instrumental in accelerating the algorithm to find optimal or near-optimal solutions. Alternatively, one might want to randomly select paths to be included within the chromosome irrespective of the rank.

Mutation

The mutation operator is aimed at increasing the diversity of solutions by perturbing each newly generated member. Similar to the other operators, the mutation operator is able to maintain feasibility of the solutions. Mutation is defined here such that the last path and corresponding flow (i.e. the last gene in the chromosome) is replaced with a set of randomly generated paths for covering the same amount of flow. To mutate a child, the procedural steps provided below are performed. For each supply time t , $\{t \in S \mid b_s(t) \neq 0\}$,

1) Assume there are h paths for shipping all supply at departure time t , i.e. $P_i(t)$, $i = 1, 2, \dots, h$. Eliminate the last path $P_h(t)$ as well as the associated flow $F_h(t)$ from the solution. Set $k = h$. Return capacity to all arcs in that path.

2) Randomly choose a path that connects the source and the sink that does not exist in the current solution. Identify the units of flow $F_k(t)$ to be sent through this path by Equation 1.

3) Update the remaining capacity of all arcs in that path.

4) Update $b_s(t)$: $b_s(t) = b_s(t) - F_k(t)$. If $b_s(t) \neq 0$, set $k = k + 1$ and return to Step 2.

Evaluation/Selection

The purpose of this step is to promote better solutions by replacing less optimal solutions with good ones. The solution quality is identified by the solution's fitness. The fitness function is formulated based on the objective function (i.e. minimize total cost). For the considered problem, the fitness value is equal to the total cost incurred in shipping all supply to the sink.

Some of the popular selection schemes are proportionate, tournament, ranking, and Boltzmann selection operators. In this work, the binary tournament selection is implemented. The tournament selection is considered to be the most efficient and least prone to premature convergence of all of the selection schemes (Goldberg and Deb, 1991). Unlike proportionate selection, the tournament selection is able to handle a minimization problem directly without having to transform it into an equivalent maximization one. In addition, it can prevent the scaling problems when most of the solutions have similar fitness.

The binary tournament operator randomly chooses pairs of chromosomes for tournaments. If two chromosomes competing in a tournament have different fitness values, the one with the better fitness is chosen. If, on the other hand, these chromosomes have the same fitness, random selection is performed. A chromosome that is chosen as a result of a tournament will not compete in subsequent tournaments. The procedure continues until a desired number of solutions are attained.

An overlapping model where parents and children compete for population slots is employed. In order to preserve the best solution for each generation, the elitism technique is employed. Elitism selects the best member from the entire population and then injects such a solution into the next generation without replacement. This technique guarantees that the best solution found in each population will not be inadvertently eliminated by the selection operator.

Termination

The algorithm terminates when the optimal solution is obtained (in case that the optimal solution is known) or the stopping criteria (e.g. the number of iterations exceeds a given threshold) are met. In the latter case, the best solution in the last generation is selected as the final solution.

Figure 5.2 demonstrates the framework of the developed GA. The first step of the algorithm is to generate the initial population consisting of μ solutions. Then, a group of 2λ individuals are randomly selected from the population for crossover and mutation. After undergoing the reproduction process, λ new solutions are added into the population and the binary tournament selection is conducted on the entire population ($\mu + \lambda$) for selecting μ solutions to be inserted into the next population. The procedure continues until the termination criteria are met.

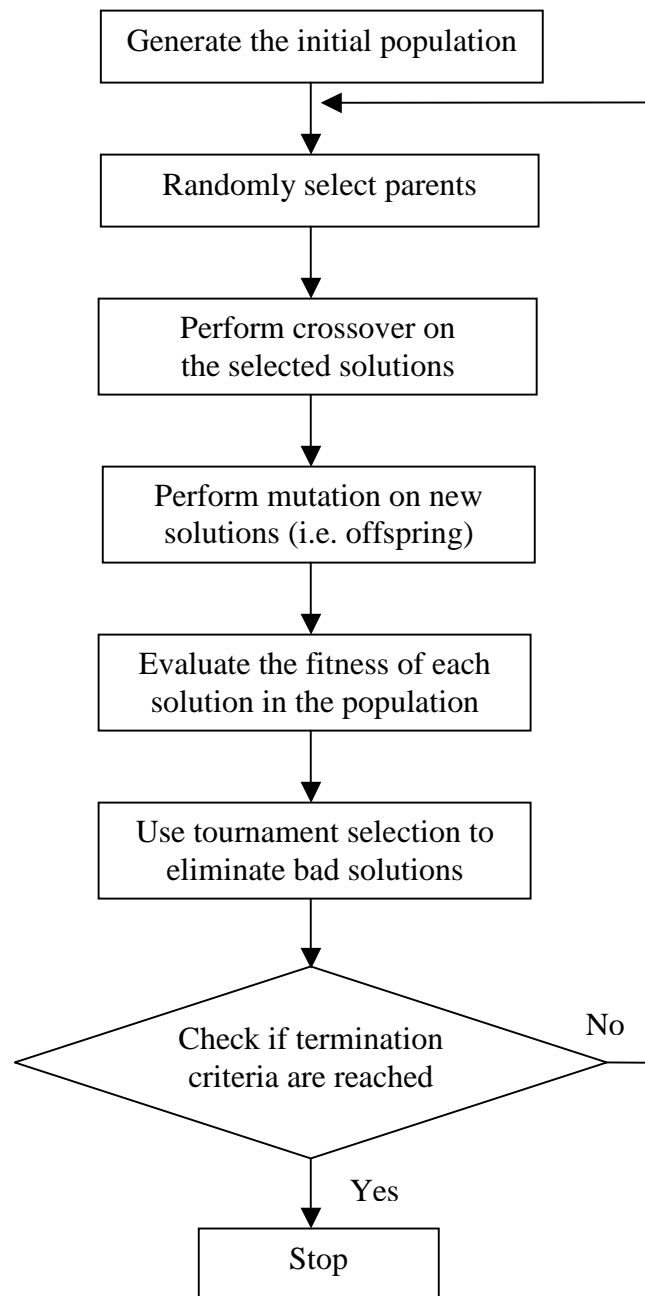


Figure 5.2. Genetic algorithm structure.

5.2.3 Illustrative Example

In this section, an example network given in Figure 5.3 is constructed to illustrate the procedure steps of the proposed GA required in completing the first generation. The network consists of 5 nodes and 8 arcs. The period of interest is discretized into three time intervals, $t = \{0, 1, 2\}$. There are 10 supply units at the source, node 1, at departure time $t = 0$, i.e. $b_1(0) = 10$. For simplicity, the travel time on each arc is assumed to be one unit of time and remains constant for the entire period. The time-varying arc costs and capacities are given in Table 5.1. The objective is to determine a pattern of flow for shipping all supply units from node 1 to node 5 with minimum total cost.

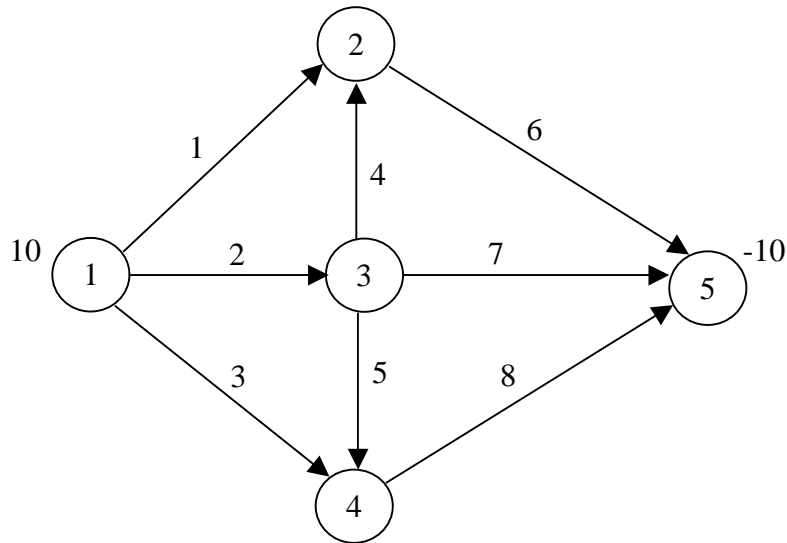


Figure 5.3. Illustrative example.

Table 5.1. Arc capacities and costs associated with each arc.

Arc	(i,j)	Capacity			Cost		
		$t = 0$	$t = 1$	$t = 2$	$t = 0$	$t = 1$	$t = 2$
1	(1,2)	5	3	4	10	14	5
2	(1,3)	3	1	3	12	5	1
3	(1,4)	4	5	2	6	8	1
4	(3,2)	1	2	3	5	2	4
5	(3,4)	4	5	2	6	3	3
6	(2,5)	2	6	6	4	6	6
7	(3,5)	4	5	9	1	7	11
8	(4,5)	2	6	6	3	3	3

Initialization

Generate initial solutions:

1) Set $i = 1$.

2) Randomly select a path $P_1(0) = (1, 6)$ from the source to the sink.

Identify the amount of flow $F_1(0) = \min(10, 5, 6) = 5$.

3) Update supply at the source: $b_1(0) = 10 - 5 = 5$.

4) Update all affected arc capacities:

$$u_1(0) = 5 - 5 = 0,$$

$$u_6(1) = 6 - 5 = 1.$$

5) Since $b_1(0) \neq 0$, set $i = 2$ and return to step 2.

2) Randomly select a path $P_2(0) = (2, 7)$ from the source to the sink.

Identify the amount of flow $F_2(0) = \min(5, 3, 5) = 3$.

3) Update supply at the source: $b_1(0) = 5 - 3 = 2$.

4) Update all affected arc capacities:

$$u_2(0) = 3 - 3 = 0,$$

$$u_7(1) = 5 - 3 = 2,$$

5) Since $b_1(0) \neq 0$, set $i = 3$ and return to step 2.

2) Randomly select a path $P_3(0) = (3, 8)$ from the source to the sink.

Identify the amount of flow $F_3(0) = \min(2, 4, 6) = 2$.

3) Update supply at the source: $b_1(0) = 2 - 2 = 0$.

4) Update all affected arc capacities:

$$u_3(0) = 4 - 2 = 2,$$

$$u_8(1) = 6 - 2 = 4.$$

5) Since $b_1(0) = 0$, Stop.

The flow pattern consists of three paths: $\{(1, 6), 5\}$, $\{(2, 7), 3\}$ and $\{(3, 8), 2\}$.

Suppose three solutions are generated in the first population as shown below.

Solution I: $\{(1, 6), 5\}$, $\{(2, 7), 3\}$ and $\{(3, 8), 2\}$. Total cost = 155.

Solution II: $\{(1, 6), 5\}$, $\{(2, 4, 6), 1\}$, $\{(2, 7), 2\}$ and $\{(3, 8), 2\}$. Total cost = 156.

Solution III: $\{(2, 4, 6), 2\}$, $\{(1, 6), 4\}$, $\{(2, 5, 8), 1\}$ and $\{(3, 8), 3\}$. Total cost = 149.

Crossover

Assume solutions I and III are randomly selected for reproduction.

1) For each path in solutions I and III, determine the cost per path capacity.

2) Rank all paths starting from the one with the minimum value.

Path I: (3, 8): 9 (C/F)

Path II: (1, 6): 16

Path III: (2, 5, 8): 18

Path IV: (2, 7): 19

Path V: (2, 4, 6): 20

3) Pick paths according to the ranking starting from Path I. Determine the flow units to be sent on that path. Continue until $b_1(0) = 0$.

Upon termination, the new solution contains three paths, $\{(3, 8), 4\}$, $\{(1, 6), 5\}$, and $\{(2, 5, 8), 1\}$, resulting in the total cost of 134.

Mutation

For the newly generated child $\{(3, 8), 4\}$, $\{(1, 6), 5\}$, $\{(2, 5, 8), 1\}$, the steps given below are followed to perturb the solution.

- 1) Eliminate the last path, $P_3(0) : (2, 5, 8)$, from the solution. Set $k = 3$
- 2) Randomly choose a new path $P_3(0) = (2, 7)$ with $F_3(0) = 1$ unit.
- 3) Update supply at the source, $b_1(0)$.
- 4) Update the affected arc capacities.
- 5) Since $b_1(0) = 0$, Stop.

The solution obtained after the mutation operator is $\{(3, 8), 4\}$, $\{(1, 6), 5\}$, $\{(2, 7), 1\}$ with the total cost of 135. This new member will be called Solution IV in the population.

Evaluation/Selection

After the crossover and mutation operators, four members exist in the population, including the new member. Three tournaments of two solutions are required for selecting three solutions to be in the next population. Note that Solution IV, which has the minimum total cost, will be injected into the next generation (i.e. elitism).

After this stage, three solutions will enter to the next generation and two of them will be randomly chosen for reproduction.

Termination

The algorithm proceeds until the stopping criteria are met.

5.2.4 Experimental Results

This section presents experimental results conducted for two purposes: 1) tuning the parameters and 2) comparing the algorithm performance with exact algorithms. The proposed GA was coded in C++ and run on a DEC Alpha XP1000 professional workstation with 1 gigabyte ram and 2 gigabyte swap, running Digital 4.0E operating system, using Digital's C++ compiler. Two network topologies consisting of 25 and 100 nodes, each with 60 time intervals were employed. The same networks as used in Chapter 4 for testing the SEscape algorithm are also applied in these experiments.

5.2.4.1 Parameter Tuning

The first part of the experiments is intended to tune the parameters. Prior to implementing the GA for a given problem, several parameters have to be set, including:

- 1) The number of generations (G_i): the number of times a new population is generated.
- 2) Population size (μ): the number of solutions maintained in each generation.
- 3) Size of parents (2λ): the number of solutions selected for mating and mutation.

- 4) Probability of crossover (C_r): the probability that a pair of chromosomes will undergo crossover in moving to the next generation.
- 5) Probability of mutation (M_t): the probability that each new child will undergo a random change in the genes.

For each test, the number of generations was fixed at 10,000 and the quality of a solution is evaluated with respect to how close it is to the optimal value. The TDQFP algorithm for solving the time-dependent quickest flow problem is modified for determining optimal solutions with no waiting. Seven sizes of population were examined, including: 20, 50, 80, 200, 250, 280 and 400. The results show that the performance of the GA markedly varied with the values of the parameters and no ideal set of parameter values exists for all problem instances. The set of parameter values found to perform best on average are $\mu = 50$, $2\lambda = 26$, $C_r = 1$, and $M_t = 1$.

In general, larger population sizes lead to more population diversity and are, thus, less prone to premature convergence. However, a dilemma in population size selection occurs as additional computations are required to process larger populations. In this work, the population size of 50 was chosen to save computational effort in fitness evaluations and solution selections. The loss of diversity due to the use of a moderate population size was compensated for by implementing both crossover and mutation operators with probability 1, i.e. every pair of parents is recombined and each child must undergo mutation. This implementation permits exploration of new search spaces without having to maintain and process an excessively large number of chromosomes.

5.2.4.2 Algorithm Performance Analysis

A number of tests were conducted to examine the performance of the proposed GA. The same networks (i.e. 25 and 100 nodes with 60 time intervals) identical to those used in tuning the parameters were employed here. Two sets of supply at the source were considered, 15 and 45 units. The results from pilot tests reveal that the procedure stopped evolving somewhat early and the final solutions were far from the optimum irrespective of the parameters used. That is, the solution was quickly trapped with suboptimal solutions. Thus, adjustments to the design of the genetic operators were made to add diversity to the solutions. The genetic operators were adjusted as follows.

- 1) In every process of assigning flow on a path, instead of pushing the flow equal to the maximum capacity on that path, the amount of flow is randomly assigned, ranging from one unit to the maximum path capacity. By altering the structure of each chromosome, this adjustment enables the algorithm to explore new solution space.
- 2) The crossover operator is adjusted such that there is some possibility that paths are randomly selected from parents rather than being picked according to the cost per path capacity as discussed in Subsection 5.2.2. This option is aimed at deterring convergence to suboptimal solutions. Numerous experiments were conducted and it was found that the best probability value was 0.5.

With these adjustments, the GA could reach optimal or near-optimal solutions with fewer generations than required in the approach originally proposed.

In order to compare the GA's performance to that of exact algorithms, how close the value of the solution produced by the GA is to the optimal value is used as the performance measure. Similar to the tuning process, the number of generations was fixed at 10,000 and the best value found upon completion of the last generation is recorded. To avoid oddly chosen source and sink pairs that might skew the results, results were drawn from a single randomly selected source and a number of randomly selected sinks. The experimental results obtained from 5 randomly selected sinks for the problems with 25 nodes and 60 time intervals are demonstrated in Table 5.2.

Table 5.2. Results for network with 25 nodes and 60 time intervals.

25 nodes					
Supply = 15			Supply = 45		
Optimal	GA	% diff	Optimal	GA	% diff
325	325	0	871	924	6.085
443	444	0.226	1242	1366	9.984
472	473	0.212	1181	1223	3.556
438	448	2.283	781	809	3.585
497	527	6.036	888	929	4.617
Average		1.751	Average		5.565

From Table 5.2, the average percentages different from the optimal value derived from five randomly selected sinks are 1.751 for the problem with 15 supply units and 5.565 for the problem with 45 supply units. Additional experiments were conducted on a larger network with 100 nodes and 60 time intervals. Again, two levels of the number of supply units at the source, 15 and 45 units, were assumed. The results drawn from one randomly selected source with ten randomly selected sinks are provided in Table 5.3.

Table 5.3. Results for network with 100 nodes and 60 time intervals.

100 nodes					
Supply = 15			Supply = 45		
Optimal	GA	% diff	Optimal	GA	% diff
374	398	6.417	1603	1711	6.737
313	341	8.946	1682	1860	10.583
305	314	2.951	1716	1837	7.051
263	268	1.901	1777	1879	5.740
351	363	3.419	1792	1901	6.083
294	296	0.680	1434	1491	3.975
327	327	0	1576	1638	3.934
273	273	0	1482	1510	1.889
184	184	0	1641	1656	0.914
355	365	2.817	1459	1500	2.810
Average		2.713	Average		4.972

The results from both tables indicate that the GA could find the solution within 10 percents of the optimal value. In all these tests, three cases could reach the optimal solution and the percentages above the known optimal value range from 3 to 5 on average. It is significant to note that in most tests, further improvement in terms of the results may be gained by properly adjusting the parameters of the GA.

Finally, a set of experiments were conducted for examining the GA performance in solving the SEscape problem (addressed in Chapter 4), whose objective is to find a flow pattern that maximizes the minimum path probability of successful arrival at the sink by supply. One can notice that in the SEscape problem, the gap between the optimal value (i.e. the maximum minimum path probability) and the second best value varies with each network configuration and could be very large. For example, the best and the second best flow pattern for shipping certain supply units from a given pair of source and sink nodes may have the minimum path probabilities of successful arrival 0.5 and 0.03, respectively, leading to the difference

below the optimal value of 94%. Hence, the closeness to the optimal value may not provide a good measure for judging the GA's performance. In order to compare the results of the GA to the exact solution found by the SEscape algorithm, the cases where the minimum path probability of the optimal solution is 1 were considered and the number of times the GA can reach this value was determined.

Table 5.4. Results for the SEscape problem.

Supply = 15			
25 nodes		100 nodes	
Optimal	GA	Optimal	GA
1	1	1	0.996
1	1	1	1
1	1	1	1
1	1	1	0.989
1	1	1	1
1	1	1	1
1	0.996	1	0.911
1	1	1	0.994
1	1	1	1
1	0.997	1	0.866

As shown in Table 5.4, the GA could find the optimal solution 8 out of 10 times for the 25 node network, and 5 out of 10 times for the 100 node counterpart. For the tests that failed to achieve the optimal value, the differences between the optimal and the final values fall within a reasonable range of the optimal solution.

5.3 A NOISY GENETIC ALGORITHM FOR STOCHASTIC, TIME-VARYING NETWORKS

In Section 5.2, travel times were assumed to be deterministically known. In this section, the GA proposed in Section 5.2 is extended for use in STV capacitated

networks, where both arc travel times and capacities are discrete random variables with probability distribution functions that vary with time. Each joint realization of the random quantities is referred to as a network state. That is, a network state is a particular instance of the set of arc travel times and capacities. In such stochastic environments, the network can take on a number of discrete states, each of which results in a different outcome. One approach to evaluating the performance of a given flow pattern with respect to an objective function under such circumstances is to evaluate the flow pattern under all possible network states. The probability of a particular realization is computed by the product of the arc attribute probabilities. One can then sum the product of the performance in each state and the probability of each state over all possible states. Because the number of network states grows exponentially with the size of the network and the number of possible values each attribute can take, enumeration of all states would require enormous effort, even for a small network. For example, a network with m arcs, each of which has D_T possible travel times and D_C possible capacities, involves $(D_T \cdot D_C)^m$ possible states. Thus, technologies that require complete enumeration of all states should be avoided.

To deal with the uncertainty in arc attribute values, the concept of noisy genetic algorithms (NGAs) is employed in this work. NGAs extend GAs for use in noisy conditions. Noise in this context is considered as any factor that impedes the accurate evaluation of the fitness of a solution. These factors can come from a variety of sources, such as the use of approximate fitness functions, the use of noisy data and knowledge uncertainty in the problem characteristics. In solving the minimum cost dynamic network flow problem in stochastic settings, the fitness of a solution cannot

precisely be identified without *a priori* knowledge of the network state that will arise upon use of the solution. Under such circumstances, each realization results in a different fitness value. A flow pattern that has a low total cost under a particular realization may have a very high total cost when evaluated under another realization. Such fitness functions are called noisy fitness functions. Through the use of NGAs, complete enumeration of all network states can be avoided. Instead, only a subset (i.e. sample) of the possible realizations is employed for evaluating chromosomes in each generation.

5.3.1 Sampling Fitness Function

Sampling fitness function is a type of noisy fitness function, which reduces noise by taking the mean of multiple noisy fitness evaluations of a solution. Instead of relying on a single fitness function, each solution is evaluated on S sample sets drawn randomly from the pool of all possible network states. The overall fitness of a solution is determined by the average of the fitness evaluations for all sample sets. Based on the Central Limit Theorem, the approximation to the actual noisy function value with a sample size of n can be computed by the following equation,

$$f_{i,S}^* = \frac{1}{S} \sum_{j=1}^S f_{i,j}^* \quad (2),$$

where $f_{i,j}^*$ = the j^{th} noisy fitness evaluation of solution i .

A challenging task of developing NGAs is the evaluation of the optimal sample size for the sampling fitness function. Miller (1997) showed that while a Monte Carlo simulation modeling needs tremendous sampling, NGAs with the

sampling fitness function can find robust solutions with a relatively small number of samples in each iteration. The technique presented in Miller's work can be used to determine the sample size that maximizes GA performance without the need for experimental trial and error. An equation is proposed by Fitzpatrick and Grefenstette (1998) for determining the optimal sample size:

$$g^*(S) = \frac{T}{(\alpha + Q \cdot S) \cdot N} \quad (3),$$

where

$g^*(S)$ = ending generation,

T = total time required by a GA,

α = fixed amount of GA overhead time per solution per generation,

Q = cost of a single fitness evaluation (a sample),

S = sample size, and

N = population size.

Given a computational bound T , the optimal sample size can be identified by evaluating the ending generation from different sample sizes. To reduce the size of the sample search space, Miller (1997) established lower and upper bounds for the optimal sample size, and proposed a pruning method for removing a large segment of the potential sample sizes between the lower and upper bounds from consideration. The accuracy of the proposed technique was experimentally proved. Gopalakrishnan (2001) proposed techniques to determine the optimal sampling strategy, which required as few as 5 samples in each iteration for addressing risk-based remediation design.

In order to implement the NGA for the minimum cost dynamic flow problem in stochastic environments, the NGA framework given in Smalley (1998) can be followed. In his work, the algorithm started with the sample size (S) of 5 for the first four generations and the sample size was increased by five sample sets every four generations. At the end of generation twelve, the best four solutions from the preceding four generations (9-12) were evaluated with 500 samples. If one or more solutions met the specified criterion, the algorithm proceeded with the same sample size for four more generations until termination. Otherwise, five sample sets were added and the entire process was repeated until the maximum number of iterations was reached.

5.3.2 Sampling Design

To determine the true noisy function value, each solution should be evaluated on all possible network states. However, this would require enormous computational effort. Instead, the function value may be estimated on only a subset of the network states (each of which is referred to as a sample). The larger the number of samples (i.e. randomly generated network states) used in estimating these values, the lower the variability in the estimate. Variance reduction techniques aid in reducing the number of samples required to accurately estimate the function value while simultaneously considering a small number of samples. In addition, such methods can be used to ensure that critical, albeit unlikely, events are not excluded from consideration and that they are not given too great a significance. A technique that appropriately selects samples is required to guarantee inclusion of the extreme cases in the correct

proportion in assessing a solution. For the application considered here, it is critical that events with very small likelihood of occurrence are considered, because such events may lead to great loss of lives. If such occurrences are given too much significance, however, the solutions may be overly conservative.

In this work, the stratified sampling method is chosen to deal with the selection of sample sets. It is found that among many other techniques, stratified sampling can effectively decrease the variance of the sample mean (Bratley et al., 1987). In the stratified sampling method, the number of times each value of the random variables is sampled can be computed using equation (4) as described next.

Let $\tau_i^j(t)$ denote the j^{th} traversal time on arc i at departure time t with associated probability of occurrence, $\rho_i^j(t)$. Likewise, $u_i^j(t)$ denotes the j^{th} capacity on arc i at departure time t with associated probability of occurrence, $\beta_i^j(t)$. Given a sample of size S , the number of times $\tau_i^j(t)$ is sampled can be computed as

$$\frac{\rho_i^j(t) \cdot \sigma_i^j(t) \cdot S}{\sum_k \rho_i^k(t) \cdot \sigma_i^k(t)} \quad (4),$$

where $\sigma_i^j(t)$ is the standard deviation of the outcome of using $\tau_i^j(t)$. The rationale behind this concept is that the larger $\sigma_i^j(t)$ is, the more $\tau_i^j(t)$ should be sampled. The similar equation can be applied for the arc capacity $u_i^j(t)$.

If the standard deviations $\sigma_i^k(t)$, $k = 1, 2, \dots, j, \dots, D$ are unknown, a pilot experiment may be conducted to estimate the variance $(\sigma_i^k(t))^2$. However, the use of variance estimates cannot guarantee variance reduction (Bratley et al., 1987). Under

such circumstances, it is suggested that excluding $\sigma_i^k(t)$ from the formula may be desirable, as it reduces variance. In the problem context considered in this dissertation, the standard deviation $\sigma_i^k(t)$ is not given and determination of such values requires extensive pilot experiments, one for each random variable. Therefore, the standard deviations are eliminated from the formula in this work.

5.3.3 Constraint Handling

In the GA proposed in Section 5.2 for solving the problem in dynamic networks with deterministic, time-varying arc travel times, infeasible solutions are not permitted. When arc attributes are known deterministically, it is possible to allow only feasible solutions at intermediate stages of the algorithm through the application of specially designed operators, including crossover and mutation operators. However, this technique cannot guarantee feasibility when the arc attributes are known only with uncertainty. In such situations, a feasible solution generated from one realization may violate some of the constraints under other realizations and it may be unlikely that there exists a single solution that is feasible for all realizations. Thus, the constraint handling technique proposed here allows infeasible solutions to be retained in the population. Penalties are imposed on solutions that are infeasible for any realization. The penalty function employed in this work is described next.

Recall that each chromosome represents a set of paths and associated amount of flow on each path. To generate the initial population, each solution is generated from the network realization whose arc attributes are set to the values that have the maximum probability of occurrence. This guarantees that all solutions generated in

the first generation do not violate the constraints for the most likely realization of travel times and capacities. In noisy fitness evaluations, a number of samples (i.e. randomly selected network states) are selected from the pool of all possible network realizations for examining the fitness of each solution. Any solution that violates the constraints is then penalized by decreasing the fitness value.

The pattern of flow given in Figure 5.4 is used to describe the proposed penalty method. In the figure, four supply units are shipped from the source node A to the sink node D along arcs 1 and 2, departing from the source at time $t = 0$. The chromosome for this flow pattern is: $\{P_1(0), F_1(0)\} = \{(1, 2), 4\}$. The capacities and travel times on arcs 1 (A-B) and 2 (B-D) under the considered realization are given in the figure. For example, arc 2 has capacity of two units at time $t = 2$ and five units at time $t = 3$.

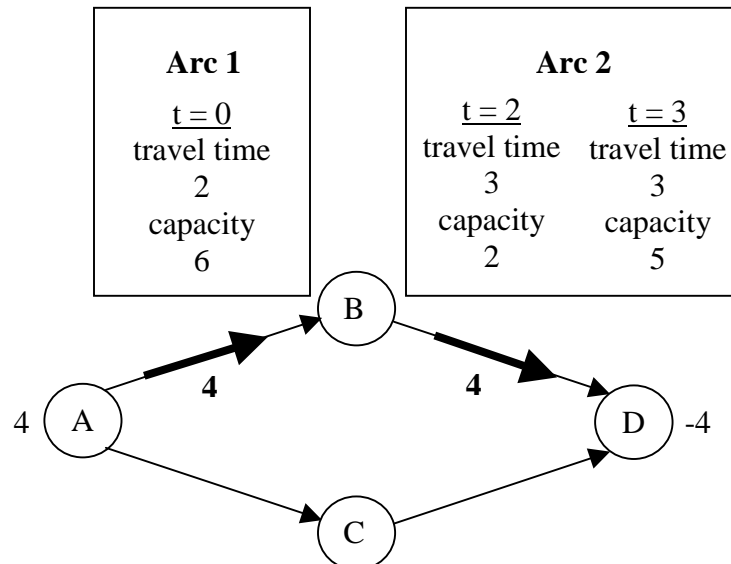


Figure 5.4. Arc attribute realization on arcs 1 and 2.

As in the deterministic case, the fitness value corresponds with the objective function value (i.e. minimum total time herein). From the given flow pattern and the network realization, all four supply units can successfully arrive at node B at time $t = 2$, incurring eight units of time in total. Upon arrival at node B, only two units can traverse arc 2, because the available capacity on this arc at departure time $t = 2$ is two. In this case, the other two units must wait at this node until the capacity on the arc is recaptured at time $t = 3$. A penalty is included in the fitness function that is equal to the cost associated with waiting at node B. For this particular example, because the fitness of a solution is the total time required to complete the shipment, the penalty is equal to the waiting time at node B for capacity recovery of arc 2. The fitness value of this pattern of flow can be computed as shown in Table 5.5.

Table 5.5. Fitness evaluation.

Action	Associated Cost (units of time)
(i) All four units traverse arc 1 at $t = 0$	$4 * 2 = 8$
(ii) Two units traverse arc 2 at $t = 2$ to arrive at node 4	$2 * 3 = 6$
(iii) Two remaining units wait at node 2 for one time interval	$2 * 1 = 2$
(iv) The two units traverse arc 2 at $t = 3$ to arrive at node 4	$2 * 3 = 6$
Fitness value	22

5.3.4 Illustrative Example

To illustrate the NGA for solving the minimum time dynamic flow problem in a STV capacitated network, a network consisting of 4 nodes and 5 arcs as portrayed in Figure 5.5 is considered. The period of interest is discretized into 10 time intervals, starting from $t = 0$. The probabilistic arc travel times and capacities are provided in Table 5.6. There are 2^{60} possible realizations (i.e. network states) of this network.

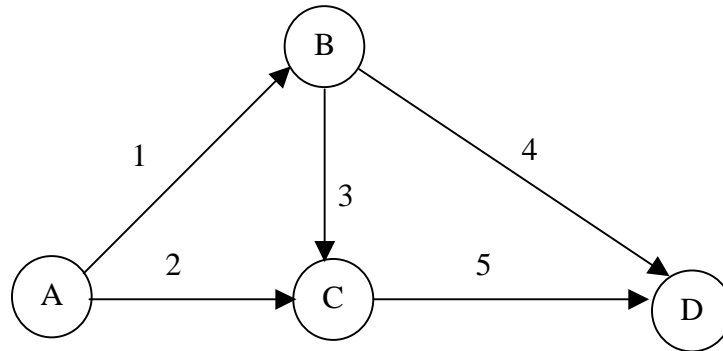


Figure 5.5. STV capacitated network.

Table 5.6. Random travel times and capacities.

Arc i	1	2	3	4	5
$\tau_i^j(t) (\rho_i^j(t))$	$t \leq 3: 1(0.3)$ 2(0.7) $t \geq 4: 4(0.8)$ 6(0.2)	$t=0: 2(0.7)$ 3(0.3) $t \geq 1: 3(0.2)$ 4(0.8)	$t \leq 1: 1(1)$ $t \geq 2: 1(0.2)$ 2(0.8)	$t \leq 1: 4(1)$ $t \geq 2: 3(0.1)$ 7(0.9)	$t \leq 2: 2(1)$ $t=3: 3(1)$ $t \geq 4: 1(0.5)$ 3(0.5)
$u_i^j(t) (\beta_i^j(t))$	$t \leq 1: 2(0.3)$ 4(0.7) $t \geq 2: 1(0.8)$ 2(0.2)	$t \geq 0: 1(1)$	$t=0: 3(0.2)$ 6(0.8) $t=1: 2(1)$ $t \geq 2: 3(1)$	$t \leq 1: 5(0.6)$ 7(0.4) $t \geq 2: 2(1)$	$t \leq 2: 2(1)$ $t \leq 4: 5(1)$ $t \geq 5: 1(0.7)$ 4(0.3)

For each arc and each time interval, the stratified sampling method is used to select the sample sets of arc travel times and capacities (i.e. the number of times each possible value of arc travel time and capacity at each departure time is selected). For example, arc 1 at time $t = 0$ has two possible traversal times, i.e. $\tau_1^1(0) = 1$ with $\rho_1^1(0) = 0.3$, and $\tau_1^2(0) = 2$ with $\rho_1^2(0) = 0.7$. Assume five samples are to be selected for evaluating the noisy fitness functions in each generation. The number of times $\tau_1^1(0)$ and $\tau_1^2(0)$ are sampled can be computed as follows:

$$\text{for } \tau_1^1(0), \frac{\rho_1^1(0) \cdot 5}{(\rho_1^1(0) + \rho_1^2(0))} = 1.5,$$

$$\text{for } \tau_1^2(0), \frac{\rho_1^2(0) \cdot 5}{(\rho_1^1(0) + \rho_1^2(0))} = 3.5.$$

In order to accommodate these sample sets, because each iteration uses five sample sets to determine the fitness value, $\tau_1^1(0)$ is randomly selected three times in every two generations. Similar procedures are employed for other arcs and departure times.

An experiment was conducted to illustrate the nature of solution on this network example with G_t (number of generations) = 100, μ (population size) = 5 and 2λ (number of parents) = 4. Assume four units are to be shipped from node A to node D. Experimental results show that the pattern of flow displayed in Figure 5.6 results in the minimum average total time of 18.67 time units.

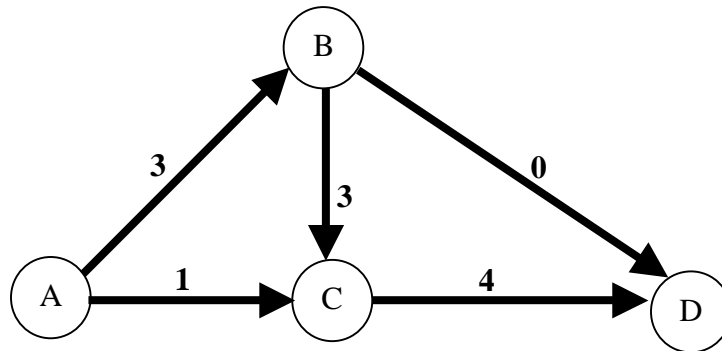


Figure 5.6. Minimum time flow.

Additional experiments were conducted on a network with 25 nodes, 99 arcs and 60 time intervals. The network is constructed from a deterministic network with some arc attributes taking on probabilistic values. In order to be able to assess optimal

solutions for each network realization, the stochastic arc attributes are designed such that only 27 network states exist. The objective is to find a pattern of flow for shipping 45 units from a randomly selected source to a randomly selected sink with minimum total time. The TDQFP algorithm was employed to determine the optimal solution for each realization. There are four optimal values found in all 27 realizations: 878, 871, 847 and 825 time units. The optimal values and the number of times they are found are provided in Table 5.7.

Table 5.7. Optimal values for 27 realizations.

Optimal value	Frequency
878	6
871	3
847	9
825	9

Because the probability of each realization can be computed from the product of the arc attribute probabilities, the expected value of the optimal value can be determined through the computed probabilities and corresponding optimal values. For this example network, the expected value is 851.68 time units. The expected value of the solution obtained from the NGA is 888 time units, leading to a difference below the optimal value of 4.26 %. This shows that the result from the NGA falls within a reasonable range of the optimal solution on average.

5.4 A NOISY GENETIC ALGORITHMS FOR MULTICRITERIA, STOCHASTIC, TIME-VARYING NETWORKS

In Section 5.3, the framework of a NGA was presented for determining the optimal flow pattern in dynamic, STV networks, where only a single criterion was considered. In general, many applications that can be modeled as network flow problems involve multiple conflicting objectives. For instance, in building evacuation, the solution to the expected flow problem may require a person to follow a path with high likelihood of failure or a path with excessively long travel time. Thus, a set of paths that considers these objectives simultaneously may be desired. Because such objectives may be conflicting in nature, the solution of such a problem will be a set of Pareto-optimal solutions. As discussed in Chapter 2, in any multiobjective problem, it is possible that all feasible solutions may be Pareto-optimal.

A host of studies have successfully used GAs to address multicriteria optimization problems. The advantages of implementing GAs on multicriteria problems are twofold. First, GAs search for optimal solutions from the entire decision spaces. Second, many potential solutions are maintained in each generation for evaluation. By combining these two features, GAs are capable of searching for a set of Pareto-optimal solutions in large, complex spaces. Commonly, a multicriteria problem is characterized by a vector of k criteria and the evaluation function consists of k attribute fitness values. There are two general approaches to solving multiobjective problems: (1) generate all Pareto-optimal solutions and select the “best compromise” solution *a posteriori* based on the decision maker’s preference function

or (2) convert the multiobjective problem to a single objective with the use of either a scalar-aggregative or an order-aggregative (non-scalar) utility function.

In the first approach, the role of GAs is to search for solutions on the efficient frontier. By performing dominance comparisons in the fitness evaluation, all dominated solutions can be discarded before the multicriteria decision-making process is made. The simplest version of this approach is to independently perform multiple single criterion searches. While taking advantage of algorithmic simplicity, the drawback of this approach lies in the lack of Pareto diversity as only extreme non-dominated solutions are sought. Other methods for addressing multiobjective problems in GAs are summarized in Bäck et al. (2000).

In the second approach, the decision maker's preference structure is represented through a utility function. Scalar-aggregative utility functions transform multiple objectives into a single scalar-valued utility function. In conjunction with GAs, such a utility function is incorporated into the fitness function for solution evaluations. A scalar fitness function is necessary for particular types of selection methods, such as roulette wheel, where the fitness values affect the probability of being selected. For order-aggregative utility functions, the decision maker provides ordinal ranking of the considered criteria. In this context, GAs are implemented such that the solution that performs best on the most important criterion is the most preferred solution. In case of ties, the next most important criterion is considered and so on. Details of one of these approaches to handling multiobjective problems are given next.

A multicriteria noisy genetic algorithm (MNGA) is presented for use in dynamic, MSTV capacitated networks with multiple criteria. It appears that no GA has been proposed in the literature for addressing optimization problems in this context. The MNGA extends the NGA framework discussed in Section 5.3 to handle multiple objectives. Herein, design of solution representation and genetic operators are the same as for the single criterion NGA. No special treatment is required for developing the encoding, crossover and mutation operators. While the proposed MNGA can be implemented with any technique as discussed previously, the order-aggregative method is illustrated here. In this method, the binary tournament selects the solution that has the higher fitness value with respect to the most important criterion to enter the next population. If there are ties, the next important criterion is considered.

The proposed MNGA is not restricted by the type or number of considered criteria. For example, several objectives may be simultaneously considered in developing evacuation instructions: minimize total time, maximize expected flow and maximize the minimum path probability of successful arrival at the sink (the SEscape problem). The MNGA was tested on the network given in Figure 5.6. Two criteria are considered: 1) the expected flow and 2) the time required for completing the shipment. The expected flow was given the highest priority. To assess the expected flow for a given flow pattern, waiting is not allowed at any intermediate node and any flow unit that cannot traverse an arc by the arrival time at that location is considered to have disappeared. The best flow pattern for shipping four units from nodes A to D is shown in Figure 5.7.

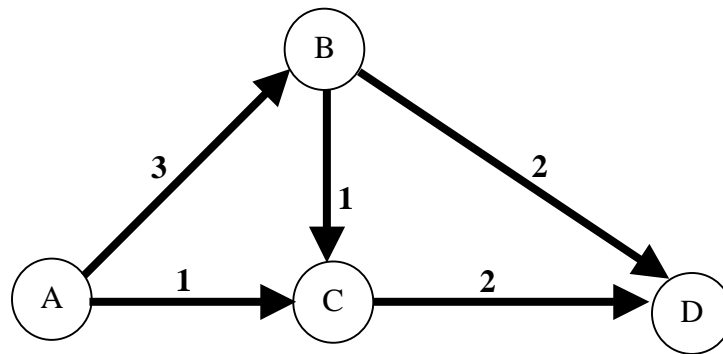


Figure 5.7. Optimal pattern of flow.

The expected flow and expected travel time of the optimal flow pattern are 3.625 units and 19.75 time units, respectively. Note that the minimum time flow shown in Figure 5.6 for the case when waiting is allowed if needed has an expected flow of 3 units and the expected travel time of 15.25 time units in this problem context.

To evaluate the performance of the MNGA on a larger network, the same network configuration (25 nodes, 99 arcs, 60 time intervals, 27 network states and 45 supply units at the source) as used in Subsection 5.3.4 was tested. The results show that the expected flow of 45 units and the minimum time of 887.6 time units are obtained. This solution means that by following the flow pattern suggested by the MNGA, all flow units can successfully arrive at the sink with the least total time.

CHAPTER 6

CONCLUSIONS AND EXTENSIONS

6.1 SYNTHESIS

This dissertation addresses optimal path problems with multiple conflicting objectives in stochastic and time-varying networks. Motivation for this work is primarily derived from two applications: providing drivers with paths in traffic networks and providing evacuees (pedestrians) with escape paths from large buildings. Exact and heuristic techniques are proposed for determining the paths for these applications. These solution approaches explicitly consider the stochastic and time-varying nature of the problem characteristics (i.e. travel times and capacities). Moreover, in capacitated networks, the fact that capacities can be recaptured over time, i.e. dynamic network flows, is recognized. Both applications involve multiple conflicting objectives. For example, in determining optimal instructions for evacuees, one may wish to maximize the likelihood that the person following a path with least probability of successful arrival at the exit and simultaneously maximize the expected number of people who will succeed in evacuating, or alternatively, minimize the total time required for all evacuees to escape. Numerous numerical experiments were conducted to assess the average performance of the proposed procedures and to illustrate the nature of the solutions that are produced.

Three exact algorithms are developed for finding adaptive path strategies when multiple conflicting objectives exist: the Adaptive Pareto-Optimal Strategy

(APS), the Adaptive Least Expected Disutility Strategy I & II (ALEDS I & II), and the Adaptive Preference Path Strategy (APPS) algorithms.

The APS algorithm determines all adaptive Pareto-optimal hyperpaths from all nodes to a destination for all departure time in MSTV networks. Given multiple criteria, such solution paths provide the traveler with the ability to dynamically choose a path to travel at each intermediate location from among all non-dominated path strategies in response to knowledge of experienced traffic conditions. While the APS algorithm does not perform well in large networks as found in the numerical experiments, such an exact procedure can be used to provide benchmark solutions on small problem instances when developing more efficient, but heuristic, approaches. Because generation of all adaptive Pareto-optimal solutions requires enormous computational effort and memory, the ALEDS algorithm is developed to efficiently generate only a single “best” hyperpath by explicitly representing the traveler’s preference structure through a linear disutility function. The ALEDS II algorithm is superior to the ALEDS I algorithm in both average run times and storage requirements. It appears that the problem of finding these solutions has not been addressed in the published literature.

While the path strategies generated by the aforementioned algorithms take into account all criteria simultaneously in the process of path selection, the APPS algorithm provides path solutions that permit the traveler to adaptively select the best path with respect to the criterion considered to be most important at each node in response to knowledge of experienced travel times on the arcs. With the ability to change preferences in this way, the traveler can adapt his or her path according to

both revealed arrival times at intermediate locations and the traveler's changing preferences. Although identical solutions can be obtained by performing the ELB algorithm (2000) multiple times, once for each criterion, the APPS algorithm offers significant computational savings as indicated by the results of numerous computational experiments.

These algorithms consider only one traveler. In many applications, however, the objective may be to determine paths for multiple travelers who will use the transport facility simultaneously. In such applications, it is necessary to consider capacity restrictions that will prevent all travelers from using the same path. For example, in evacuating a building, only a limited number of evacuees can use any particular escape route at the same moment in time. Thus, solution will involve multiple paths and assignment of travelers (or evacuees) to the paths. An exact solution approach, the SEscape algorithm, is presented to address the problem of determining the optimal *a priori* flow pattern for shipping supply units from a source to a sink in dynamic capacitated networks, where arc travel times are time-varying and arc capacities are stochastic and time-varying (STV). The algorithm is motivated by the need to determine optimal escape paths for evacuees seeking refuge from a large burning building or a building that has come under attack. The SEscape algorithm takes into account issues of fairness among the evacuees. Specifically, it seeks the pattern of flow that maximizes the minimum path probability of successful arrival of supply at the sink. Following the solutions provided by the SEscape algorithm guarantees that the likelihood that any person who is subject to the greatest risk will succeed in escaping is maximized. Results of numerical experiments show

that the SEscape algorithm performs significantly better than its worst-case computational complexity. The algorithm can be used in evacuation planning, enabling safer evacuation of a building in the event of military attack, fire, natural disaster, or other circumstances warranting quick escape.

While the SEscape problem assumes STV capacities, arc travel times are deterministic quantities. For addressing the problem of determining *a priori* path flow decisions to ship supply from a source to a sink such that multiple objectives are achieved, a meta-heuristic based on the principles of noisy genetic algorithm (NGA) is presented. First, the framework for the genetic algorithm (GA) is proposed for determining minimum cost flow pattern in a dynamic network, where arc travel times are deterministic, time-dependent. The proposed GA framework can be extended for use with many other objectives, e.g. minimize total time, maximize expected flow or maximize the minimum path probability of successful arrival at the sink (the SEscape problem). A specialized encoding scheme and genetic operators are developed to handle the complexity of path flow solutions and constraints. The results of numerical experiments on several different network configurations show that, on average, the proposed GA could find the solution within 5 percent of the optimal value.

The GA framework is also extended to address the problem in more complex STV and MSTV capacitated networks that cannot be efficiently solved by currently existing exact techniques. The concept of noisy fitness functions is employed to evaluate the fitness value of each chromosome under uncertain conditions. The multicriteria noisy genetic algorithm (MNGA) is developed for use in MSTV capacitated networks. The proposed MNGA is not restricted by the type or number of

considered criteria. For example, several objectives may be simultaneously considered in determining the optimal flow pattern: minimize total time, maximize expected flow and maximize the minimum path probability of successful arrival at the sink (the SEscape problem). Two generic approaches for extending the NGA for use in solving multicriteria dynamic network flow problems with STV arc attributes (including arc capacities) are proposed. The first approach seeks to generate all of the Pareto-optimal solutions and the second approach reduces the problem to a single objective problem by employing a utility function to represent a decision-maker's preference structure. For illustration purposes, details of a preemptive method from the second class of approaches are provided. In the preemptive method, the criteria are ranked according to their importance to the decision-maker and the solution that optimizes the most important criterion is given greater preference. The preemptive method implementation of the proposed framework is tested on two networks. The results indicate that near-optimal solutions can be obtained with this approach.

6.2 FUTURE EXTENSIONS

While the procedures proposed herein show promise, there are some areas one may consider for enhancement. Potential future extensions are given next.

The Adaptive Pareto-optimal Strategy Problems

The APS algorithm proposed in Chapter 2 for generating all Pareto-optimal hyperpaths requires enormous computational effort. Upon termination of the algorithm, a large number of Pareto-optimal hyperpaths may exist at each node and

departure time, each of which maintains the expected values of all criteria. Therefore, enormous memory storage may be required to obtain all non-dominated solutions. A procedure that seeks only a subset of Pareto-optimal hyperpaths would be beneficial.

The ALEDS algorithm relies on a linear disutility function to assess a single “best” hyperpath solution in MSTV networks. A utility function provides a formal, mathematical representation of the decision maker’s (DM) preference structure. The associated weights represent the relative importance among criteria, and thus, have a direct influence on the decision that will be made. In order to select a single preferred path in a multicriteria path problem regardless of the technique employed, the preference structure of the traveler must be properly elicited. In reality, a traveler’s preferences are not fixed, nor can they be represented by a single model form. Instead, they fluctuate over time in response to knowledge of travel conditions as a driver travels through the network. For example, the importance of travel time might wane once it is revealed that most of the streets between the traveler’s current location and desired destination are subject to the same undesirable level of congestion as observed on the current path. Under such circumstances, the traveler’s preferences may change and the model used to represent the traveler’s preferences will need to be updated. Hence, updating a model of the traveler’s preferences (e.g. adjusting criterion weights in a utility function representation) is necessary to assure that the most desired path will be properly chosen. The ALEDS algorithm requires the use of a utility function that is monotonic and additive. In many real-world applications, however, the traveler’s preference structure may not be represented by a

linear function. A future extension may be to develop solution approaches that can handle less restrictive preference functions.

Once the preferences are explored, the adjusted model of the traveler's preferences will be used to determine updated adaptive hyperpaths. The simple but time-consuming method for determining a new solution is to resolve from scratch, independent of any prior computations. To avoid unnecessarily excessive computation time, a faster reoptimization method that relies on only preceding solution paths and the updated traveler preferences would be beneficial.

The SEscape Problem

The SEscape algorithm is developed for determining the optimal set of *a priori* path flows in dynamic networks with STV arc capacities. As found through the numerical experiments, the MPP algorithm, which is used as a subroutine within the SEscape algorithm, consumes significant portion of the computational time required by the SEscape. This is because the MPP algorithm performs unnecessary tasks by determining the MPP from all nodes to a destination for all departure times. One approach to improve the complexity of the SEscape algorithm is to develop a more efficient approach that finds only a single path from the source to the sink at a particular time to substitute for the MPP algorithm. In addition, the SEscape algorithm assumes arc travel times to be known deterministically. However, many applications require consideration of STV arc travel times. Because only a heuristic is proposed in this dissertation for use in such networks, the development of exact techniques is also an interesting extension for future research.

Network Flow Problems in MSTV Capacitated Networks

A genetic algorithm is proposed for solving the problem of determining optimal path flows in MSTV capacitated networks. The performance of the heuristic is evaluated through experiments on small networks for illustrating the nature of the solutions. Additional tests on larger networks would need to be conducted for further evaluation. Additionally, only the preemptive approach is implemented in the proposed NGA for addressing multicriteria problems. The effectiveness of the NGA implemented with other concepts, such as generating Pareto-optimal solutions or using a utility function, should be investigated. Moreover, future research may be the development of techniques for determining the exact solution or bounds on the solution of these multi-objective, stochastic and time-varying problems in capacitated networks.

APPENDICES

APPENDIX A ILLUSTRATIVE EXAMPLE FOR THE APS ALGORITHM

This section is designed to clarify the essential steps of the APS algorithm and to provide insight into the nature of the solutions. A simple example problem composed of 5 nodes and 7 arcs as shown in Figure A.1 is constructed for computational illustration. Table A.1 contains the relevant arc attribute data.

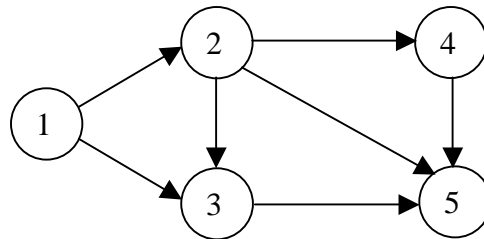


Figure A.1. Illustrative example

Table A.1. Probabilistic time and cost data.

Travel Time							
Time	arc (1,2)	arc (1,3)	arc (2,3)	arc (2,4)	arc (2,5)	arc (3,5)	arc (4,5)
$t=0$	1(0.8)*	2(1.0)	1(0.9)	3(0.9)	2(0.8)	3(0.2)	1(0.8)
	2(0.2)		3(0.1)	4(0.1)	6(0.2)	5(0.8)	2(0.2)
$t=1$	2(0.3)	1(0.8)	2(0.8)	2(0.8)	5(0.9)	2(0.5)	1(0.9)
	4(0.7)	2(0.2)	3(0.2)	4(0.2)	7(0.1)	4(0.5)	3(0.1)
$t=2$	1(0.8)	3(0.3)	1(0.9)	1(0.3)	3(0.2)	1(0.8)	1(0.9)
	2(0.2)	4(0.7)	3(0.1)	2(0.7)	6(0.8)	2(0.2)	2(0.1)
$t=3$	1(0.5)	1(0.8)	3(0.5)	3(0.3)	7(0.5)	3(0.5)	4(0.3)
	3(0.5)	2(0.2)	4(0.5)	4(0.7)	8(0.5)	4(0.5)	7(0.7)
$t=4$	2(0.8)	2(1.0)	2(0.8)	1(0.5)	6(0.9)	5(0.8)	5(0.5)
	3(0.2)		4(0.2)	3(0.5)	7(0.1)	6(0.2)	6(0.5)
Cost							
$t=0$	3(0.8)	7(0.9)	3(0.8)	1(0.2)	7(0.3)	8(0.2)	4(0.8)
	4(0.2)	9(0.1)	4(0.2)	3(0.8)	8(0.7)	9(0.8)	5(0.2)
$t=1$	4(0.5)	10(0.9)	4(0.5)	2(0.9)	6(0.5)	7(0.3)	5(0.9)
	5(0.5)	11(0.1)	5(0.5)	4(0.1)	8(0.5)	8(0.7)	6(0.1)
$t=2$	2(0.8)	7(0.8)	3(0.9)	2(0.9)	6(0.5)	7(0.3)	3(0.4)
	3(0.2)	9(0.2)	4(0.1)	4(0.1)	8(0.5)	8(0.7)	7(0.6)
$t=3$	3(0.8)	5(0.1)	3(0.8)	2(0.5)	7(0.3)	5(0.9)	2(0.1)
	4(0.2)	7(0.9)	4(0.2)	3(0.5)	8(0.7)	6(0.1)	4(0.9)
$t=4$	3(0.9)	8(0.5)	3(0.9)	2(0.5)	6(0.5)	4(0.4)	7(0.3)
	4(0.1)	9(0.5)	5(0.1)	3(0.5)	8(0.5)	8(0.6)	8(0.7)

*arc attribute (associated probability)

The period of interest is discretized into five time intervals, $S = \{0, \dots, 4\}$. Suppose there are two criteria considered in the hyperpath selection: travel time (criterion 1) and travel cost (criterion 2), respectively. Because the computational process for solving this simple example is large and repetitive, this section presents only a portion of the entire process. This portion of the computational steps of determining the complete set of Pareto-optimal hyperpaths from all origins to node 5 for each departure time in the peak period is shown next.

Initialize the elements of the vector labels and path pointers. $SE = \{5\}$.

Iteration 1

Scan node 5. $SE = \{ \}$.

$j = 5, i \in \{2, 3, 4\}$.

For $i = 2$,

$t = 0$

for $Q = \{(1,1), (2,1)\}$,

$$\begin{aligned} \eta_2^1(0) &= \sum_{(z_1, x) \in Q} [c_{25}^{1z_1}(0) + \lambda_{5,x}^1(0 + c_{25}^{1z_1}(0))] \cdot \rho_{25}^{1z_1}(0) \\ &= (2+0) \cdot 0.8 + (6+0) \cdot 0.2 = 2.8. \end{aligned}$$

$$\begin{aligned} \eta_2^2(0) &= \sum_{(z_1, x) \in Q} \sum_{z_2=1}^2 [c_{25}^{2z_2}(0) + \lambda_{5,x}^2(0 + c_{25}^{1z_1}(0))] \cdot \rho_{25}^{1z_1}(0) \cdot \rho_{25}^{2z_2}(0) \\ &= (7+0) \cdot 0.8 \cdot 0.3 + (8+0) \cdot 0.8 \cdot 0.7 + (7+0) \cdot 0.2 \cdot 0.3 + (8+0) \cdot 0.2 \cdot 0.7 = 7.7. \end{aligned}$$

$$\lambda_{21}(0) = (2.8, 7.7).$$

Insert 1 into $X_2(0)$. Set $\pi_{21}(0) = 5$ and $q_{21}(0) = \{(1,1), (2,1)\}$. $SE = \{2\}$.

-
- (Continue to loop over t .)
-

For $i = 3$ and 4 and each $t \in \mathcal{S}$, compute the label components and associated pointers.

Check dominance and make necessary updates. Within this iteration, at least one component of the labels at nodes 2, 3 and 4 has been updated. Thus, $SE = \{2,3,4\}$ at

the end of this iteration. The following figure shows the labels $\lambda_{iv}(t)$, $\pi_{iv}(t)$ and

$q_{iv}(t)$ associated with nodes 2, 3, and 4, respectively, at the end of this iteration.

Time	Node 2	Node 3	Node 4
	Position 1	Position 1	Position 1
0	$(2.8,7.7)^3$ 5, $\{(1,1), (2,1)\}$	$(4.6,8.8)$ 5, $\{(1,1), (2,1)\}$	$(1.2,4.2)$ 5, $\{(1,1), (2,1)\}$
1	$(5.2,7)$ 5, $\{(1,1), (2,1)\}$	$(3,7.7)$ 5, $\{(1,1), (2,1)\}$	$(1.2,5.1)$ 5, $\{(1,1), (2,1)\}$
2	$(5.4,7)$ 5, $\{(1,1), (2,1)\}$	$(1.2,7.7)$ 5, $\{(1,1), (2,1)\}$	$(1.1,5.4)$ 5, $\{(1,1), (2,1)\}$
3	$(7.5,7.7)$ 5, $\{(1,1), (2,1)\}$	$(3.5,5.1)$ 5, $\{(1,1), (2,1)\}$	$(6.1,3.8)$ 5, $\{(1,1), (2,1)\}$
4	$(6.1,7)$ 5, $\{(1,1), (2,1)\}$	$(5.2,6.4)$ 5, $\{(1,1), (2,1)\}$	$(5.5,7.7)$ 5, $\{(1,1), (2,1)\}$

Figure A.2. Solutions for iteration 1.

Iteration 2

Scan node 4. $SE = \{2,3\}$.

$j = 4, i \in \{2\}$.

For $i = 2$ and each $t \in \mathcal{S}$, compute the label components and associated pointers.

Check dominance and make necessary updates. The following figure shows the updates associated with node 2 at the end of this iteration.

³ $\lambda_{21}(0) = (2.8,7.7)$, $\pi_{21}(0) = 5$, $q_{21}(0) = \{(1,1), (2,1)\}$

Time	Node 2	
	Position 1	Position 2
0	(2.8,7.7) 5, {(1,1), (2,1)}	(9.14,6.79) 4, {(1,1), (2,1)}
1	(5.2,7) 5, {(1,1), (2,1)}	(8.38,6.78) 4, {(1,1), (2,1)}
2	(5.4,7) 5, {(1,1), (2,1)}	
3	(7.5,7.7) 5, {(1,1), (2,1)}	
4	(6.1,7) 5, {(1,1), (2,1)}	

Figure A.3. Solutions for iteration 2.

Iteration 3

Scan node 3. $SE = \{2\}$.

$j = 3, i \in \{1,2\}$.

For $i = 1$ and each $t \in \mathcal{S}$, compute the label components and associated pointers.

Check dominance and make necessary updates.

For $i = 2$ and each $t \in \mathcal{S}$, compute the label components and associated pointers.

Check dominance and make necessary updates.

The following figure shows the updates associated with nodes 1 and 2 at the end of this iteration. $SE = \{1,2\}$.

Time	Node 1		Node 2	
	Position 1		Position 1	Position 2
0	(3.2,14.9)	3, {(1,1)}	(2.8,7.7)	(9.14,6.79)
			5, {(1,1),(2,1)}	4, {(1,1),(2,1)}
1	(2.86,17.28)	3, {(1,1),(2,1)}	(5.2,7)	(8.38,6.78)
			5, {(1,1),(2,1)}	4, {(1,1),(2,1)}
2	(8.9,13.8)	3, {(1,1),(2,1)}	(5.4,7)	(4.87,8.33)
			5, {(1,1),(2,1)}	3, {(1,1),(2,1)}
3	(6.4,13.2)	3, {(1,1),(2,1)}	(7.5,7.7)	
			5, {(1,1),(2,1)}	
4	(7.2,14.9)	3, {(1,1)}	(6.1,7)	
			5, {(1,1),(2,1)}	

Figure A.4. Solutions for iteration 3.

Iteration 4

Scan node 2. $SE = \{1\}$.

$j = 2, i \in \{1\}$.

For $i = 1$,

$t = 0$

There are two possible travel times on arc (1,2) at departure time $t = 0$,

i.e. $z_1 = \{1,2\}$. $c_{12}^{11}(0) = 1$ and $\rho_{12}^{11}(0) = 0.8$, $c_{12}^{12}(0) = 2$ and $\rho_{12}^{12}(0) = 0.2$.

At node 2, there are two Pareto-optimal labels at time $0 + c_{12}^{11}(0)$ and $0 + c_{12}^{12}(0)$,

i.e. $X_2(0 + c_{12}^{11}(0)) = X_2(1) = \{1,2\}$ and $X_2(0 + c_{12}^{12}(0)) = X_2(2) = \{1,2\}$.

Then, four different Q sets of (z_1, x) pairs would be generated as follows:

Set 1: (1,1), (2,1)

Set 2: (1,1), (2,2)

Set 3: (1,2), (2,1)

Set 4: (1,2), (2,2)

for set 1: $Q = \{(1,1), (2,1)\}$,

$$\begin{aligned}\eta_1^1(0) &= \sum_{(z_1,x) \in Q} [c_{12}^{1z_1}(0) + \lambda_{2x}^1(0 + c_{12}^{1z_1}(0))] \cdot \rho_{12}^{1z_1}(0) \\ &= (1+5.2) \cdot 0.8 + (2+5.4) \cdot 0.2 = 6.44.\end{aligned}$$

$$\begin{aligned}\eta_1^2(0) &= \sum_{(z_1,x) \in Q} \sum_{z_2=1}^2 [c_{12}^{2z_2}(0) + \lambda_{2x}^2(0 + c_{12}^{1z_1}(0))] \cdot \rho_{12}^{1z_1}(0) \cdot \rho_{12}^{2z_2}(0) \\ &= (3+7) \cdot 0.8 \cdot 0.8 + (4+7) \cdot 0.8 \cdot 0.2 + (3+7) \cdot 0.2 \cdot 0.8 + (4+7) \cdot 0.2 \cdot 0.2 = 10.2.\end{aligned}$$

$\eta_1(0)$ is non-dominated, thus $\lambda_{12}(0) = (6.44, 10.2)$.

Insert 2 into $X_1(0)$. Set $\pi_{12}(0) = 2$ and $q_{12}(0) = \{(1,1), (2,1)\}$. $SE = SE \cup \{1\}$.

for set 2: $Q = \{(1,1), (2,2)\}$,

$$\begin{aligned}\eta_1^1(0) &= \sum_{(z_1,x) \in Q} [c_{12}^{1z_1}(0) + \lambda_{2x}^1(0 + c_{12}^{1z_1}(0))] \cdot \rho_{12}^{1z_1}(0) \\ &= (1+5.2) \cdot 0.8 + (2+4.87) \cdot 0.2 = 6.334.\end{aligned}$$

$$\begin{aligned}\eta_1^2(0) &= \sum_{(z_1,x) \in Q} \sum_{z_2=1}^2 [c_{12}^{2z_2}(0) + \lambda_{2x}^2(0 + c_{12}^{1z_1}(0))] \cdot \rho_{12}^{1z_1}(0) \cdot \rho_{12}^{2z_2}(0) \\ &= (3+7) \cdot 0.8 \cdot 0.8 + (4+7) \cdot 0.8 \cdot 0.2 + (3+8.33) \cdot 0.2 \cdot 0.8 + (4+8.33) \cdot 0.2 \cdot 0.2 \\ &= 10.466.\end{aligned}$$

$\eta_1(0)$ is non-dominated, thus $\lambda_{13}(0) = (6.334, 10.466)$.

Insert 3 into $X_1(0)$. Set $\pi_{13}(0) = 2$ and $q_{13}(0) = \{(1,1), (2,2)\}$.

for set 3: $Q = \{(1,2), (2,1)\}, \dots$

for set 4: $Q = \{(1,2), (2,2)\}, \dots$

-
- (Continue to loop over t .)
-

Continue in the same manner until the SE list is empty when step 2 is first called.

The final hyperpath solutions for every node and time interval are provided in Figure A.5.

Time	Node 1			
	Position 1	Position 2	Position 3	Position 4
0	(3.2,14.9) 3, {(1,1)}	(6.44,10.2) 2, {(1,1),(2,1)}	(6.334,10.466) 2, {(1,1),(2,2)}	(8.984,10.024) 2, {(1,2),(2,1)}
1	(2.86,17.28) 3, {(1,1),(2,1)}	(9.92,11.71) 2, {(1,1),(2,1)}		
2	(8.42,9.76) 2, {(1,1),(2,1)}			
3	(6.4,13.2) 3, {(1,1),(2,1)}	(8.1,10.2) 2, {(1,1),(2,1)}		
4	(7.2,14.9) 3, {(1,1)}	(8.3,10.1) 2, {(1,1),(2,1)}		

Time	Node 2		Node 3	Node 4
	Position 1	Position 2	Position 1	Position 1
0	(2.8,7.7) 5, {(1,1),(2,1)}	(9.14,6.79) 4, {(1,1),(2,1)}	(4.6,8.8) 5, {(1,1), (2,1)}	(1.2,4.2) 5, {(1,1), (2,1)}
1	(5.2,7) 5, {(1,1),(2,1)}	(8.38,6.78) 4, {(1,1),(2,1)}	(3,7.7) 5, {(1,1), (2,1)}	(1.2,5.1) 5, {(1,1), (2,1)}
2	(5.4,7) 5, {(1,1),(2,1)}	(4.87,8.33) 3, {(1,1),(2,1)}	(1.2,7.7) 5, {(1,1), (2,1)}	(1.1,5.4) 5, {(1,1), (2,1)}
3	(7.5,7.7) 5, {(1,1),(2,1)}		(3.5,5.1) 5, {(1,1), (2,1)}	(6.1,3.8) 5, {(1,1), (2,1)}
4	(6.1,7) 5, {(1,1),(2,1)}		(5.2,6.4) 5, {(1,1), (2,1)}	(5.5,7.7) 5, {(1,1), (2,1)}

Figure A.5. Pareto-optimal hyperpath solutions.

Upon termination of the algorithm, $\lambda_{11}(0) = (3.2,14.9)$, $\lambda_{12}(0) = (6.44,10.2)$,

$\lambda_{13}(0) = (6.334,10.466)$ and $\lambda_{14}(0) = (8.984,10.024)$ with associated hyperpath

pointers, $\pi_{11}(0) = 3$ and $\pi_{12}(0) = \pi_{13}(0) = \pi_{14}(0) = 2$, respectively. The hyperpath from node 1 at departure time $t = 0$ indicates that there are two Pareto-optimal moves the traveler can take: head to either node 2 or node 3. If node 2 is chosen and the arrival time at this node is $t = 1$, two Pareto-optimal moves are suggested: go to node 5 directly or go to node 4 followed by node 5. If, on the other hand, the arrival time at node 2 is $t = 2$, going to node 5 directly and going to node 3 followed by node 5 are both considered to be efficient. If the traveler departs node 1 and chooses to head to node 3, the suggested move is to go from node 3 directly to node 5, regardless of the actual arrival time at node 3. These solution hyperpaths from node 1 to node 5 departing from node 1 at $t = 0$ are portrayed in Figure A.6.

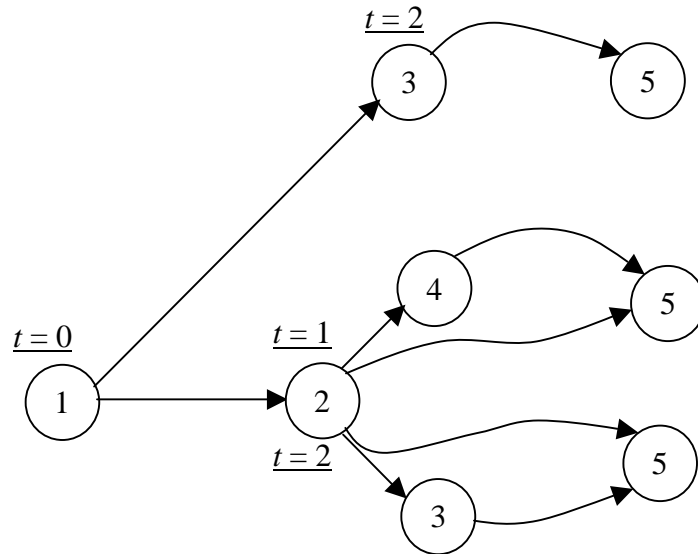


Figure A.6. Pareto-optimal hyperpaths from node 1 to node 5 at departure time 0.

There are four *a priori* paths: 1-3-5, 1-2-4-5, 1-2-5 and 1-2-3-5. The expected travel times and expected travel costs, respectively, for these *a priori* paths when departing from node 1 at $t = 0$ are as follows.

Path 1-3-5: (3.2,14.9)

Path 1-2-4-5: (9.38,10.37)

Path 1-2-5: (6.44,10.2)

Path 1-2-3-5: (7.006,12.754)

There are two *a priori* Pareto-optimal paths: paths 1-2-5 and 1-3-5. Both of these paths are also Pareto-optimal hyperpath solutions for the adaptive problem.

APPENDIX B MATHEMATICAL FORMULATION OF THE SESCAPE PROBLEM

The mathematical formulation of the SEscape algorithm for the example network given in Figure 4.9 can be written as follows.

$$\text{Max [min } \{ (P_{12}^{x_{12}(0)}(0) \cdot P_{24}^{x_{24}(2)}(2)), (P_{12}^{x_{12}(0)}(0) \cdot P_{23}^{x_{23}(2)}(2) \cdot P_{34}^{x_{34}(4)}(4)), \\ (P_{13}^{x_{13}(0)}(0) \cdot P_{34}^{x_{34}(4)}(4)), (P_{14}^{x_{14}(0)}(0)) \}]$$

Subject to

$$x_{12}(0) + x_{13}(0) + x_{14}(0) = 3.$$

$$x_{23}(2) + x_{24}(2) - x_{12}(0) = 0$$

$$x_{34}(4) - x_{13}(0) - x_{23}(2) = 0$$

$$-x_{14}(0) - x_{24}(2) - x_{34}(4) = -3$$

$$0 \leq x_{12}(0) \leq 4$$

$$0 \leq x_{13}(0) \leq 1$$

$$0 \leq x_{14}(0) \leq 3$$

$$0 \leq x_{23}(2) \leq 2$$

$$0 \leq x_{24}(2) \leq 2$$

$$0 \leq x_{34}(4) \leq 2$$

REFERENCES

1. C. C. Aggarwal, R. K. Ahuja, J. Hao, J. B. Orlin, Diagnosing Infeasibilities in Network Flow Problems, *Mathematical Programming*, Vol. 81, 1998, pp. 263-280.
2. R. Ahuja, T. Magnanti, J. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1993, Chap. 4-5.
3. R. K. Ahuja, J. B. Orlin, P. Sharma, P. T. Soddalingam, A Network Simplex Algorithm with $O(n)$ Consecutive Degenerate Pivots, *Operations Research Letters*, Vol. 30, 2002, pp. 141-148.
4. E. J. Anderson, A. B. Philpott, *Optimisation of Flows in Networks Over Time*, Probability, Statistics and Optimisation, John Wiley & Sons Ltd, Chichester, New York, 1994, Chap. 27.
5. J. E. Aronson, A Survey of Dynamic Network Flows, *Annals of Operations Research*, Vol. 20, 1989, pp. 1-66.
6. T. Bäck, D. Fogel, Z. Michalewicz, *Handbook of Evolutionary Computation*, IOP Publishing Ltd. and Oxford University Press, 2000.
7. D. Bertsekas, F. Guerriero, R. Musmanno, Parallel Asynchronous Label-Correcting Methods for Shortest Paths, *Journal of Optimization Theory and Applications*, Vol. 88, 1996, pp. 297-320.
8. P. Bratley, B. L. Fox, L. E. Schrage, *A Guide to Simulation*, 2nd ed., Springer-Verlag, New York, 1987.
9. R. E. Burkard, K. Dlaska, B. Klinz, The Quickest Flow Problem, *ZOR-Methods and Models of Operations Research*, Vol. 37, 1993, pp. 31-58.

10. X. Cai, D. Sha, C. K. Wong, Time-Varying Minimum Cost Flow-Problems, *European Journal of Operational Research*, Vol. 131, 2001a, pp. 352-374.
11. X. Cai, D. Sha, C. K. Wong, Time-Varying Universal Maximum Flow Problems, *Mathematical and Computer Modelling*, Vol. 33, 2001b, pp. 407-430.
12. H. I. Calvete, Network Simplex Algorithm for the General Equal Flow Problem, *European Journal of Operational Research*, Vol. 150(3), 2003, pp. 585-600.
13. H. I. Calvete, The Quickest Path Problem with Interval Lead Times, *Computers & Operations Research*, Vol. 31, 2004, pp. 383-395.
14. M. Carey, C. Hendrickson, Bounds on Expected Performance of Networks with Links Subject to Failure, *Networks*, Vol. 14, 1984, pp. 439-456.
15. R. L. Carraway, T. L. Morin, Theory and Applications of Generalized Dynamic Programming: An Overview, *Computers and Operations Research*, Vol. 16, No. 10/11, 1988, pp. 779-788.
16. R. L. Carraway, T. L. Morin, H. Moskowitz, Generalized Dynamic Programming for Multicriteria Optimization, *European Journal of Operational Research*, Vol. 44, 1990, pp. 95-104.
17. I. Chabini, Discrete Dynamic Shortest Path Problems in Transportation Applications: Complexity and Algorithms with Optimal Run Time, *Transportation Research Record*, No. 1645, 1998, pp. 170-175.
18. L. Chalmet, R. Francis, P. Saunders, Network Models for Building Evacuation, *Management Science*, Vol. 28, 1982, pp. 86-105.
19. T. S. Chang, L. K. Nozick, M. A. Turnquist, Multi-Objective Path-Finding in Stochastic Dynamic Networks, with Application to Routing Hazardous Materials

- Shipments, forthcoming in *Transportation Science*.
20. Y. L. Chen, Y. H. Chin, The Quickest Path Problem, *Computers & Operations Research*, Vol. 17, 1990, pp. 153-161.
 21. R. K. Cheung, Iterative Methods for Dynamic Stochastic Shortest Path Problems, *Naval Research Logistics*, Vol. 45, 1998, pp. 769-789.
 22. W. Choi, H. W. Hamacher, S. Tufekci, Modeling of Building Evacuation Problems by Network Flows with Side Constraints, *European Journal of Operational Research*, Vol. 35, 1988, pp. 98-110.
 23. J. C. N. Climaco, E. Q. V. Martins, A Bicriterion Shortest Path Algorithm, *European Journal of Operational Research*, Vol. 1982, No. 11, PP.399-404.
 24. K. L. Cooke, E. Halsey, The Shortest Route Through a Network with Time-Dependent Internodal Transit Times, *Journal of Mathematical Analysis and Applications*, Vol. 14, 1966, pp. 493-498.
 25. H. W. Corley, I. D. Moon, Shortest Path in Networks with Vector Weights, *Journal of Optimization Theory and Applications*, Vol. 46, No. 1, 1985, pp.79-86.
 26. J. M. Coutinho-Rodrigues, J. C. N. Clímaco, J. R. Current, An Interactive Bi-Objective Shortest Path Approach: Searching for Unsupported Nondominated Solutions, *Computers and Operations Research*, Vol. 26, 1999, pp. 789-798.
 27. T. J. Cova, J. P. Johnson, A Network Flow Model for Lane-Based Evacuation Routing, *Transportation Research Part A*, Vol. 37(7), 2003, pp. 579-604.
 28. N. D. Curet, J. DeVinney, M. E. Gaston, An Efficient Network Flow Code for Finding All Minimum Cost S-T Cutsets, *Computers and Operations Research*, Vol. 29, 2002, pp. 205-219.

29. C. Davies, P. Lingras, Genetic Algorithms for Rerouting Shortest Paths in Dynamic and Stochastic Networks, *European Journal of Operational Research*, Vol. 144, 2003, 27-38.
30. R. B. Dial, A Model and Algorithm for Multicriteria Route-Mode Choice, *Transportation Research Part B*, Vol. 13B, 1979, pp. 311-316.
31. S. E. Dreyfus, An Appraisal of Some Shortest-Path Algorithms, *Operation Research*, Vol. 17, 1969, pp. 395-412.
32. A. Eiger, P. B. Mirchandani, H. Soroush, Path Preferences and Optimal Paths in Probabilistic Networks, *Transportation Science*, Vol. 19, No. 1, 1985, pp. 75-84.
33. J. M. Fitzpatrick, J. J. Grefenstette, Genetic Algorithms in Noisy Environments, *Machine Learning*, Vol. 3, 1998, pp. 101-120.
34. L. K. Fleischer, Universally Maximum Flow with Piecewise-Constant Capacities, *Networks*, Vol. 38(3), 2001, pp. 115-125.
35. L. Fleischer, É. Tardos, Efficient Continuous-Time Dynamic Network Flow Algorithms, *Operations Research Letters*, Vol. 23, 1998, pp. 71-80.
36. L. R. Ford, D. R. Fulkerson, A Suggested Computation for Maximal Multi-Commodity Network Flows, *Management Science*, Vol. 5, 1958, pp. 97-101.
37. L. R. Ford, D. R. Fulkerson, *Flows in Networks*, Princeton University, Princeton, NJ, 1962.
38. R. L. Francis, A "Uniformity Principle" for Evacuation Route Allocation, *Journal of Research of the National Bureau of Standards*, Vol. 86, No. 5, 1981, pp. 509-513.
39. O. Frank, W. Gaul, On Reliability in Stochastic Graphs, *Networks*, Vol. 12, 1982,

- pp. 119-126.
40. L. Fu, An Adaptive Routing Algorithm for In-Vehicle Route Guidance Systems with Real-Time Information, *Transportation Research Part B*, Vol. 35, 2001, pp. 749-765.
 41. L. Fu, L. R. Rilett, Expected Shortest Paths in Dynamic and Stochastic Traffic Networks, *Transportation Research Part B*, Vol. 32, No. 7, 1998, pp. 499-516.
 42. G. Gallo, G. Longo, S. Pallottino, S. Nguyen, Directed Hypergraphs and Applications, *Discrete Applied Mathematics*, Vol. 42, 1993, pp. 177-201.
 43. G. Gallo, S. Pallottino, Shortest Path Methods: A Unifying Approach, *Mathematical Programming Study* 26, North-Holland, Amsterdam, 1986, pp. 38-64.
 44. M. Gen, R. Cheng, S. S. Oren, Network Design Techniques Using Adapted Genetic Algorithms, *Advances in Engineering Software*, Vol. 32, 2001, pp. 731-744.
 45. G. D. Glockner, G. L. Nemhauser, A Dynamic Network Flow Problem with Uncertain Arc Capacities: Formulation and Problem Structure, *Operations Research*, Vol. 48, No. 2, March-April 2000, pp. 233-242.
 46. G. D. Glockner, G. L. Nemhauser, C. A. Tovey, Dynamic Network Flow with Uncertain Arc Capacities: Decomposition Algorithm and Computational Results, *Computational Optimization and Applications*, Vol. 18, 2001, pp. 233-250.
 47. D. E. Goldberg, *Genetic Algorithms in Search. Optimization and Machine Learning*, Addison-Wesley, New York, 1989
 48. D. E. Goldberg, K. Deb, A Comparative Analysis of Selection Schemes Used in

- Genetic Algorithms, In Rawlins, G. J. E. (Ed.), *Foundations of Genetic Algorithms*, San Mateo, CA: Morgan Kaufmann, 1991, pp. 69-93.
49. G. Gopalakrishnan, *Optimal Sampling in a Noisy Genetic Algorithm for Risk-Based Remediation Design*, M. S. Thesis, University of Illinois, 2001.
50. R. Hall, *The Fastest Path through a Network with Random Time-Dependent Travel Times*, *Transportation Science*, Vol. 20, No. 3, 1986, pp. 182-188.
51. J. Halpern, *A Generalized Dynamic Flows Problem*, *Networks*, Vol. 9, 1979, pp. 133-167.
52. W. Hamacher, S. Tufekci, *On the Use of Lexicographic Min Cost Flows in Evacuation Modeling*, *Naval Research Logistics*, Vol. 34, 1987, pp. 487-503.
53. P. Hansen, *Bicriterion Path Problems*, in: G. Fandel, T. Gal, Eds, *Multiple Criteria Decision Making: Theory and Applications*, *Lectures Notes in Economics and in Mathematical Systems* 177 (Springer, Heidelberg), 1980, pp. 109-127.
54. M. I. Henig, *The Shortest Path Problem with Two Objective Functions*, *European Journal of Operational Research*, Vol. 25, 1985, pp. 281-291.
55. B. Hoppe, E. Tardos, *The Quickest Transshipment Problem*, *Mathematics of Operations Research*, Vol. 25, No. 1, February 2000, pp. 36-62.
56. J. J. Jarvis, H. D. Ratliff, *Some Equivalent Objectives for Dynamic Network Flow Problems*, *Management Science*, Vol. 28, No. 1, January 1982, pp. 106-109.
57. J. P. Jarvis, D. R. Shier, *An Improved Algorithm for Approximating the Performance of Stochastic Flow Networks*, *INFORMS Journal on Computing*, Vol. 8, No. 4, Fall 1996, pp. 355-360.
58. R. Jentsch, *Reliability Analysis of Flow Networks*, *Advances in Stochastic*

- Models for Reliability, Quality and Safety, W. Kahle, E. Collani, J. Franz and U. Jensen (eds), Birkhäuser, Boston, 1998, pp. 235-245.
59. D. F. Jones, S. K. Mirrazavi, M. Tamiz, Multi-Objective Meta-Heuristics: An Overview of the Current State-of-the-Art, *European Journal of Operational Research*, Vol. 137, 2002, pp. 1-9.
60. C. Karbowicz, J. MacGregor Smith, A K-Shortest Path Routing Heuristic for Stochastic Evacuation Networks, *Engineering Optimization*, Vol. 7, 1984, pp. 253-280.
61. D. E. Kaufman, R. L. Smith, Fastest Paths in Time-Dependent Networks for Intelligent Vehicle-Highway Systems Application, *IVHS Journal*, Vol. 1(1), 1993, pp. 1-11.
62. A. M. Law, W. D. Kelton, *Simulation Modeling and Analysis*, 3rd ed., McGraw-Hill, 2000.
63. YK. Lin, A Simple Algorithm for Reliability Evaluation of a Stochastic-Flow Network with Node Failure, *Computers and Operations Research*, Vol. 28, 2001, pp. 1277-1285.
64. YK. Lin, Using Minimal Cuts to Evaluate the System Reliability of a Stochastic-Flow Network with Failures at Nodes and Arcs, *Reliability Engineering and System Safety*, Vol. 75, 2002a, pp. 41-46.
65. YK. Lin, Two-Commodity Reliability Evaluation for a Stochastic-Flow Network with Node Failure, *Computers and Operations Research*, Vol. 29, 2002b, pp. 1927-1939.

66. YK. Lin, Extend the Quickest Path Problem to the System Reliability Evaluation for a Stochastic-Flow Network, *Computers and Operations Research*, Vol. 30, 2003, pp. 567-575.
67. R. P. Loui, Optimal Paths in Graphs with Stochastic or Multidimensional Weights, *Communications of the ACM*, Vol. 26, 1983, pp. 670-676.
68. C. Lucet, J. Manouvrier, *Statistical and Probabilistic Models in Reliability*, D.C. Lonescu and N. Limnios (eds), Birkhäuser, Boston, 1999, pp. 279-294.
69. E. Q. V. Martins, On a Multicriteria Shortest Path Problem, *European Journal of Operational Research*, Vol. 16, 1984, pp. 236-245.
70. B. L. Miller, *Noise, Sampling, and Efficient Genetic Algorithms*, Ph.D. Dissertation, University of Illinois, 1997.
71. E. Miller-Hooks, Least-Expected Time Paths with Recourse in Stochastic, Time-Varying Transportation and Data Networks, *Networks*, Vol. 37, No. 1, 2001, pp. 35-52.
72. E. Miller-Hooks, T. Krauthammer, Intelligent Evacuation, Rescue and Recovery Concept, Proceedings of the 30th United States Department of Defense Explosives Safety Seminar, Atlanta, Georgia, August 2002.
73. E. Miller-Hooks, H. Mahmassani, Least Possible Time Paths in Stochastic, Time-Varying Networks, *Computers and Operations Research*, Vol. 25, No. 12, 1998a, pp. 1107-1125.
74. E. Miller-Hooks, H. Mahmassani, Optimal Routing of Hazardous Materials in Stochastic, Time-Varying Transportation, *Transportation Research Record*, No. 1645, 1998b, pp. 143-151.

75. E. Miller-Hooks, H. Mahmassani, Least Expected Time Paths in Stochastic, Time-Varying Transportation Networks, *Transportation Science*, Vol. 34, No. 2, 2000, pp. 198-215.
76. E. Miller-Hooks, H. Mahmassani, Path Comparisons for A Priori and Time-Adaptive Decisions in Stochastic, Time-varying Networks, *European Journal of Operational Research*, Vol. 146, No. 1, 2003, pp. 67-82.
77. E. Miller-Hooks, S. S. Patterson, On Solving Quickest Time Problems in Time-Dependent, Dynamic Networks, *Journal of Mathematical Modeling and Algorithms*, Vol. 3(1), 2004, pp. 39-71.
78. E. Minieka, Maximal, Lexicographic, and Dynamic Network Flows, *Operations Research*, Vol. 21, 1973, pp. 517-527.
79. P. B. Mirchandani, H. Soroush, Optimal Paths in Probabilistic Networks: A Case with Temporary Preferences, *Computers and Operations Research*, Vol. 12, No. 4, 1985, pp. 365-381.
80. P. B. Mirchandani, M. M. Wiecek, Routing with Nonlinear Multiattribute Cost Functions, *Applied Mathematics and Computation*, Vol. 54, 1993, pp. 215-239.
81. P. Modesti, A. Sciomachen, A Utility Measure for Finding Multiobjective Shortest Paths in Urban Multimodal Transportation Networks, *European Journal of Operational Research*, Vol. 111, 1998, pp. 495-508.
82. B. J. T. Morgan, *Elements of Simulation*, Chapman & Hall, London, 1984.
83. J. Mote, I. Murthy, D. L. Olson, A Parametric Approach to Solving Bicriterion Shortest Path Problems, *European Journal of Operational Research*, Vol. 53, 1991, pp. 81-92.

84. T. Munakata, D. J. Hashier, A Genetic Algorithm Applied to the Maximum Flow Problem, The Fifth International Conference on Genetic Algorithms 1993, Urbana-Champaign, IL, July 17-22, 1993, pp. 488-493.
85. I. Murthy, S. Her, Solving Min-Max Shortest-Path Problems on a Network, Naval Research Logistics, Vol. 39, 1992, pp. 669-683.
86. I. Murthy, D. L. Olson, An Interactive Procedure using Domination Cones for Bicriterion Shortest Path Problems, European Journal of Operational Research, Vol. 72, 1994, pp. 417-431.
87. I. Murthy, S. Sarkar, A Relaxation-Based Pruning Technique for a Class of Stochastic Shortest Path Problems, Transportation Science, Vol. 30, No. 3, 1996, pp. 220-236.
88. H. Nagamochi, T. Ibaraki, Maximum Flows in Probabilistic Networks, Networks, Vol. 21, 1991, pp. 645-666.
89. N. Nagy, S. G. Akl, The Maximum Flow Problem: A Real-Time Approach, Parallel Computing, Vol. 29, 2003, pp. 767-794.
90. L. R. Nielsen, K. A. Andersen, D. Pretolani, Bicriterion Shortest Hyperpaths in Random Time-Dependent Networks, IMA Journal of Management Mathematics, Vol. 14, No. 3, 2003, pp. 271-303.
91. L. K. Nozick, G. F. List, M. A. Turnquist, Integrated Routing and Scheduling in Hazardous Materials Transportation, Transportation Science, Vol. 31, No. 3, 1997, pp. 200-215.
92. S. Opananon, E. Miller-Hooks, Multicriteria Adaptive Hyperpaths in Stochastic, Time-Varying Networks, Submitted for possible publication in European Journal

- of Operational Research.
93. A. Orda, R. Rom, Shortest-Path and Minimum-Delay Algorithms in Networks with Time-Dependent Edge-Length, *Journal of the Association for Computing Machinery*, Vol. 37, No. 3, 1990, pp. 607-625.
 94. A. Orda, R. Rom, On Continuous Network Flows, *Operations Research Letters*, Vol. 17, 1995, pp. 27-36.
 95. J. B. Orlin, A Faster Strongly Polynomial Minimum Cost Flow Algorithm, *Operations Research*, Vol. 41, No. 2, 1993, pp. 338-350.
 96. S. Pallottino, Shortest-Path Methods: Complexity, Interrelations and New Propositions, *Networks* Vol. 14, 1984, pp. 257-267.
 97. U. Pape, Implementation and Efficiency of Moore-algorithms for the Shortest Route Problem, *Mathematical Programming*, Vol. 7, 1974, pp. 212-222.
 98. M. Pióro, Á. Szentesi, J. Harmatos, A. Jüttner, P. Gajowniczek, S. Kozdrowski, On Open Shortest Path First Related Network Optimization Problems, *Performance Evaluation*, Vol. 48, 2000, pp. 201-223.
 99. G. H. Polychronopoulos, J. N. Tsitsiklis, Stochastic Shortest Path Problems with Recourse, *Networks*, Vol. 27, No. 2, 1996, pp. 133-143.
 100. D. Pretolani, A Directed Hypergraph Model for Random Time Dependent Shortest Paths, *European Journal of Operational Research*, Vol. 123, 2000, pp. 315-324.
 101. J. S. Provan, A Polynomial-Time Algorithm to Find Shortest Path with Recourse, *Networks*, Vol. 41, No. 2, 2003, pp. 115-125.
 102. J. B. Rosen, S. Z. Sun, G. L. Xue, Algorithms for the Quickest Path Problem

- and the Enumeration of Quickest Paths, *Computers & Operations Research*, Vol. 18, 1991, pp. 579-584.
103. A. W. Sadek, B. L. Smith, M. J. Demetsky, Dynamic Traffic Assignment Genetic Algorithms Approach, *Transportation Research Record*, Vol. 1588, 1997, pp. 95-103.
104. A. Sedeno-Noda, C. González-Martín, An Algorithm for the Biobjective Integer Minimum Cost Flow Problem, *Computers and Operations Research*, Vol. 28, 2001, pp. 139-156.
105. J. B. Smalley, Risk-Based In Situ Bioremediation Design, M. S. Thesis, University of Illinois, 1998.
106. P. T. Sokkalingam, R. K. Ahuja, J. B. Orlin, New Polynomial-Time Cycle-Canceling Algorithms for Minimum-Cost Flows, *Networks*, Vol. 36(1), 2000, pp. 53-63.
107. B. S. Stewart, C. C. White, Three Solution Procedures for Multiobjective Path Problems, *Control-Theory and Advanced Technology*, Vol.5, No. 4, 1989, pp. 443-470.
108. G. S. Sulijoadikusumo, L. K. Nozick, Multiobjective Routing and Scheduling of Hazardous Materials Shipments, *Transportation Research Record*, No. 1613, 1998, pp. 96-104.
109. K. Talebi, J. MacGregor Smith, Stochastic Network Evacuation Models, *Computers & Operations Research*, Vol. 12, No. 6, 1985, pp. 559-577.
110. M. A. Turnquist, Routes, Schedules and Risks in Transporting Hazardous Materials, *Strategic Planning in Energy and Natural Resources*, North-Holland,

- Amsterdam, 1987, pp. 289-302.
111. G. A. Vignaux, Z. Michalewicz, A Genetic Algorithm for the Linear Transportation Problem, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 21, No. 2, 1991, pp. 445-452.
 112. S. T. Waller, A. K. Ziliaskopoulos, On the On-Line Shortest Path Problem with Limited Arc Cost Dependencies, *Networks*, Vol. 40, No. 4, 2002, pp. 216-227.
 113. A. Warburton, Approximation of Pareto Optima in Multiple-Objective, Shortest-Path Problems, *Operations Research*, Vol. 35, No. 1, 1987, pp. 70-79.
 114. A. B. Wijeratne, M. A. Turnquist, P. B. Mirchandani, Multiobjective Routing of Hazardous Materials in Stochastic Networks, *European Journal of Operational Research*, Vol. 65, 1993, pp. 33-43.
 115. W. L. Wilkinson, An Algorithm for Universal Maximal Dynamic Flows in a Network, *Operations Research*, Vol. 19, 1971, pp. 1602-1612.
 116. T. Yamada, A Network Flow Approach to a City Emergency Evacuation Planning, *International Journal of Systems Science*, Vol. 27, No. 10, 1996, pp. 931-936.
 117. A. K. Ziliaskopoulos, Optimum Path Algorithms on Multidimensional Networks: Analysis, Design, Implementation and Computational Experience, Ph.D. Thesis, University of Texas at Austin.
 118. A. K. Ziliaskopoulos, H. Mahmassani, Time-Dependent, Shortest-Path Algorithm for Real-Time Intelligent Vehicle Highway System Applications, *Transportation Research Record*, No. 1408, 1993, pp. 94-100.

119. K. G. Zografos, C. F. Davis, Multi-Objective Programming Approach for Routing Hazardous Materials, *Journal of Transportation Engineering*, Vol. 115, No. 6, 1989, pp. 661-673.

Vita

Sathaporn Opananon, the son of Sunthorn and Laeiat Opananon, was born in Bangkok, Thailand on November 21, 1975. Sathaporn received his Bachelor of Engineering with Second Class Honors from Chulalongkorn University in Bangkok, Thailand, in May of 1997. In 1998, he entered a competitive examination arranged by the Royal Thai Government and was granted the scholarship for pursuit of Master's degree in the United States. Sathaporn received the Master of Science in Civil Engineering, majoring in transportation from the Pennsylvania State University in May of 2000. In August 2000, Sathaporn chose to stay with his advisor, Dr. Elise Miller-Hooks in the Pennsylvania State University to pursue a Ph.D. In January 2001, Dr. Elise took a leave from academia, which sparked Sathaporn's move to the Department of Civil and Environmental Engineering at the University of Illinois at Urbana-Champaign. In August 2001, Sathaporn and his advisor reunited at the Pennsylvania State University. In August 2003, Sathaporn transferred to the University of Maryland to continue his pursuit of the Ph.D. in transportation with his long-lasting advisor, Dr. Elise.

Permanent address: 1827-1829 Phaholyothin Rd., Ladyao, Chatuchak, Bangkok 10900, Thailand.

This dissertation was typed by the author.