Routing in Probabilistic Networks

by

Jonathan Ramsay Key

B.S. Mathematics, B.S. Computer Science, Tufts University (1998)

Submitted to the Department of Civil and Environmental Engineering
and the Operations Research Center
in partial fulfillment of the requirements for the degrees of
Master of Science in Transportation
and
Master of Science in Operations Research
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2004

Author .....
Department of Civil and Environmental Engineering and the
Operations Research Center
April 28, 2004

Certified by .....
William Weinstein
Charles Stark Draper Laboratory
Thesis Supervisor

Certified by .....
Cynthia Barnhart
Professor of Civil and Environmental Engineering
Thesis Supervisor

Accepted by .....
James B. Orlin
Codirector, Operations Research Center

Accepted by .....
Heidi Nepf
Chairman, Department Committee on Graduate Students

[This page intentionally left blank.]

Routing in Probabilistic Networks

by

Jonathan Ramsay Key

Submitted to the Department of Civil and Environmental Engineering and the
Operations Research Center
on April 28, 2004, in partial fulfillment of the
requirements for the degrees of
Master of Science in Transportation
and
Master of Science in Operations Research

Abstract

This thesis considers the problem of routing in a network where the travel times along the arcs are modeled as independent random variables. A standard approach to routing in such networks is to select a path with the least expected travel time. One of the problems with this approach is that it does not take into consideration factors such as the travel time variance. Additionally, such an approach implicitly assumes each user in the network has the same routing objective.

In this thesis we develop an approach to routing in probabilistic networks in which these problems are addressed. The fundamental concept in our approach is that, for a given user with a set of routing options at a given node, we approximate the distributions of travel time for these options. Using these approximate distributions, the options are compared according to a user-specified routing objective, and the best option is selected. The primary benefit of this approach is that one is not limited to a particular routing objective as the computed distributions of travel time allow us to efficiently determine an effective routing option for a arbitrary routing objective that depends on factors of random travel time other than the mean.

The distribution of travel time adopted in this thesis is the *minimum* travel time probability distribution, which is the distribution of travel time over all fastest paths. In a class of networks termed as series-parallel networks, the minimum travel time distribution can be calculated efficiently. For general, non-series-parallel networks, the approximation we adopt is the minimum travel time distribution obtained from a related series-parallel network.

The performance and the benefits of this approach to routing are illustrated on three networks. The numerical results are obtained using an efficient implementation of the algorithms proposed in this thesis. We also consider the problem of generating an acyclic graph from a cyclic graph, and we propose a data structure that allows for the efficient calculation of the sum and minimum of independent random variables.

Thesis Supervisor: William Weinstein
Title: Charles Stark Draper Laboratory

Thesis Supervisor: Cynthia Barnhart
Title: Professor of Civil and Environmental Engineering

[This page intentionally left blank.]

Acknowledgments

I would like to thank Professor Cindy Barnhart of the M.I.T. Center for Transportation and Logistics, and Bill Weinstein and Steve Kolitz of The Charles Stark Draper Laboratory for making my graduate education possible. Bill also served as my Draper advisor and it has been a pleasure working with him.

I would also like to thank my M.I.T. advisor, Professor Ismail Chabini. Professor Chabini is a dedicated researcher and an excellent teacher.

Graduate school would have been very difficult without the support and friendship of my peers at M.I.T. and Draper. I am proud to have worked with such excellent people.

Finally, my work would not have been possible without the love and support of my wife, Kiri, and my family.

This thesis was prepared at The Charles Stark Draper Laboratory, Inc., under Internal Company Sponsored Research Project 13177.

Publication of this thesis does not constitute approval by Draper or the sponsoring agency of the findings or conclusions contained herein. It is published for the exchange and stimulation of ideas.

Jonathan Ramsay Key

[This page intentionally left blank.]

Contents

List of Figures

List of Tables

[This page intentionally left blank.]

List of Algorithms

[This page intentionally left blank.]

Chapter 1

Introduction

This thesis presents an approach to routing in probabilistic networks. In this chapter, we discuss the problem, our motivation, and the merits of modeling networks as probabilistic networks. We then introduce a routing procedure for making routing decisions in probabilistic networks. We conclude this chapter with a summary of the contributions and an outline of this thesis.

1.1 The Problem

This thesis considers the problem of routing in a network where the travel times along arcs are modeled as independent random variables. We refer to such a network as a probabilistic network. When travel time is deterministic, a user will typically select the path with the least travel time. However, in probabilistic networks, the meaning of "least travel time" is not clear in a deterministic sense since the path with the least travel time depends on the realizations of the arc travel times, and an optimal solution may include multiple paths rather than a single path.

A standard approach is to interpret the term "least travel time" as the least expected travel time. Although this approach can be useful, there are at least two problems with it. First, it does not take into account critical properties of the travel time such as variance. For example, a user might prefer a path with a *higher* expected travel time in exchange for a *lower* variance in travel time. Second, such an approach implicitly assumes each user in the network has the same routing objective. While this assumption is convenient for analysis, in practice different users naturally have different routing objectives that they would like to optimize. While a conservative user might select the path that is fastest on average, another user might prefer the path with the fastest possible travel time (regardless of the probability of this travel time being realized).

This thesis develops an approach to routing in probabilistic networks that addresses these issues. The fundamental concept in our approach is that we focus on the computation of the probability distributions of travel times, rather than being restricted to a particular routing objective.

1.2 The Motivation

Before we elaborate on our routing problem, we discuss the practical motivation behind it. Consider a transportation network where a user at location s wants to travel to location t. At the highest level, we can separate the factors affecting user travel time into two categories. The first category is composed of decisions made and controlled by the user, while the second category is composed of all uncontrollable external influences such as incidents, road conditions, and traffic jams. It is these external influences that provide the primary difficulty in accurately assessing user travel time.

If we could obtain accurate information about these external influences *a priori*, then determining the optimal path would be trivial, as any deterministic shortest path algorithm could be used to calculate such a path. Similarly, if it is possible to obtain updated information en route, we could use this information to re-optimize path selection. Unfortunately, building an infrastructure with such capabilities would be difficult and costly, if at all possible. Furthermore, even if such an infrastructure existed, it is not clear whether such information could be delivered in a timely and effective manner.

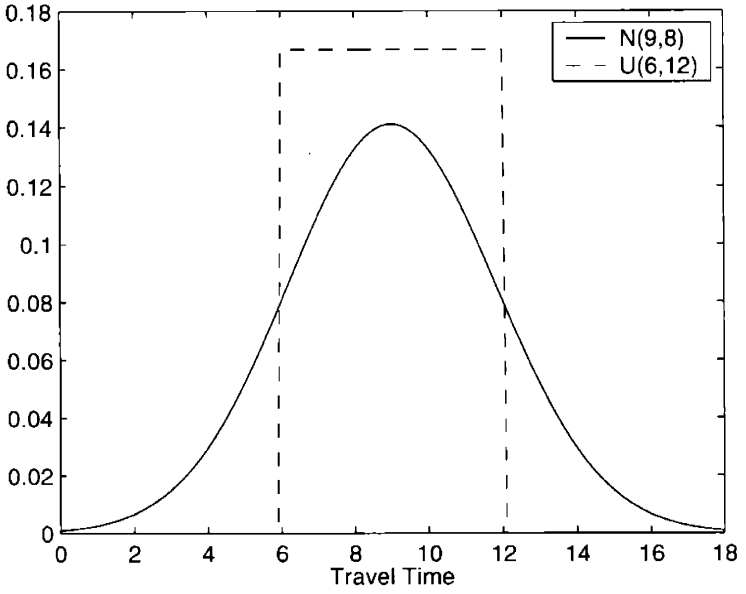
An alternate way to address the external influences on travel time is to use a model of travel time. Naturally, we would like a model that closely resembles reality. Therefore, such a model should be able to encompass the inherent randomness in travel time. This suggests that modeling travel time as a random variable is an appropriate choice. In other words, travel times take on different values with different probabilities. Using this approach, the transportation network now becomes a probabilistic transportation network, where the travel time along each arc in the network is modeled as random variable.

1.3 Routing with Random Variables

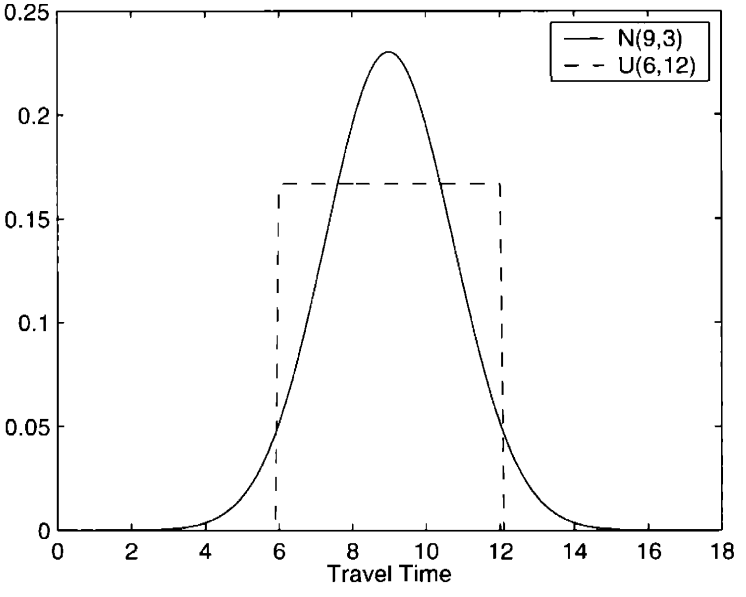
In terms of making routing decisions, the use of random variables to model travel time creates several problems. The primary difficulty is that it is not clear what constitutes a valid routing objective since the path with the least travel time depends on the realizations of the arc travel time random variables. Although users typically want to select the path with the "least travel time", this term is ambiguous when travel times are modeled as random variables.

To clarify this ambiguity, we consider some possible interpretations of "least travel time". One possible interpretation is the least expected travel time. This is a straightforward measure and it also makes sense that we might want to take a path that, on average, leads to the least travel time. In this case, consider the probability density functions in Figure 1-1(a). Let each probability density function represent the travel time for a particular path from location s to location t. The notation $\mathcal{N}(9, 8)$ refers to a normal probability density function with mean 9 and variance 8. The notation $\mathcal{U}(6, 12)$ refers to a uniform probability density function from 6 to 12.

Each random variable has an expected value of 9. But which path is better?



(a) The Travel Time Density Functions of Two Different Paths from Location s to Location t with the Same Expectation. $\mathcal{N}(9, 8)$ and $\mathcal{U}(6, 12)$.



(b) The Travel Time Density Functions of Two Different Paths from Location s to Location t with the Same Expectation and Variance. $\mathcal{N}(9, 3)$ and $\mathcal{U}(6, 12)$.

Figure 1-1: Comparing the Travel Time of Two Different Paths.

Which one will get the user to location t in the "least travel time"? The answer is not straight-forward. Since the expected travel times are equal, perhaps the user would prefer the path with the least variance. That is, the path with travel time distributed according to $\mathcal{U}(6, 12)$ would be selected. On the other hand, if the user wanted the path with the least *possible* travel time, the path with travel time distributed according to $\mathcal{N}(9, 8)$ would be selected. If the expected values were not equal, a user might trade a higher expected travel time for a lower variance in travel time. Another routing objective might be to have a certain level of *confidence* (expressed as a probability) that the trip will take less than 7 time units, in which case the path with travel time $\mathcal{N}(9, 8)$ would be selected.

Finally, to further complicate the notion of "least travel time", consider Figure 1-1(b) where the travel time random variables have the same expectation and variance. Which path should a user choose? It is not clear.

1.4 Routing Objectives

The previous section illustrates that, with random travel times, there may be several different, yet valid, routing objectives in traveling from location s to location t. For a given user with a particular set of routing options, we note that if we knew the travel time distribution for each option *a priori*, then the user could specify *any* routing objective, and we could efficiently determine the best routing option for the user. This is the motivation behind the fundamental concept in this thesis; rather than analyzing a particular routing objective, we focus on travel time distributions.

Let us define the term *actual* travel time distribution as the probability distribution of the travel time that a user will *actually* experience. In other words, the actual travel time distribution will be the travel time distribution of the path that the user actually travels on. There are several difficulties in calculating the actual travel time distribution. First, we need to enumerate and calculate the travel time distributions of all possible paths, which cannot be accomplished efficiently. More importantly, by definition, the actual travel time distribution only becomes known *a posteriori*; after a full routing specification is made including the user-specified routing objective, the relevant paths, and any other related constraints. This is a significant problem because, as we have noted, there are several valid routing objectives in probabilistic networks. Additionally, this is in conflict with our motivation to have the user specify a routing objective *after* the travel time distributions of the routing options have been calculated.

Although the calculation of the actual travel time distribution is awkward and problematic, there is a related probability distribution that is a useful approximation of the actual travel time distribution. Let us define this distribution as the *minimum* travel time distribution. This distribution is the distribution of the travel time over all possible fastest paths. One favorable consequence of using the minimum travel time distribution for routing is that, since it is "composed" of the travel time over all possible fastest paths, it implicitly "imposes" the natural routing objective of finding the fastest path.

It turns out that the general problem of calculating the minimum travel time distribution is $\mathcal{NP}$-Hard ([45],[41], see Section 2.6). Consequently, this problem is usually simplified by replacing the travel time random variables with deterministic values. The classic choice is to replace each arc travel time random variable with its expectation.

An alternate approach takes advantage of another property of the minimum travel time distribution: in the case of a *series-parallel* network, the calculation of the minimum travel time distribution can be made exactly and efficiently [26]. Furthermore, in this case, the expected value of the minimum travel time distribution is a *lower-bound* for the expected value of the actual travel time distribution (see Chapter 4). This suggests that the minimum travel time distribution is at least a good first-order approximation of the actual travel time distribution.

For a non-series-parallel network, we approximate the minimum travel time distribution by calculating the minimum travel time distribution on a related series-parallel network. This related series-parallel network is generated from the original non-series-parallel network by conditioning on the travel times of certain arcs. Furthermore, the expected value of this approximated minimum travel time distribution is a *lower-bound* for the expected value of the actual travel time distribution (see Chapter 4). The series-parallel property is the fundamental property that is exploited in this thesis.

1.5 An Approach to Routing in Probabilistic Networks

Given that a user arrives at location i and is destined for location t, there are two basic approaches to routing. The first approach is to select an entire path to location t *a priori*. The second approach is to select a "neighbor" of location i to go to next (rather than a whole path). An example of the latter approach is the primary step of the Bellman-Ford routing algorithm, which is:

$$\hat{d}_{it} = \min_{j \in N^+(i)} (d_{ij} + \hat{d}_{jt})$$

where $N^+(i)$ is the set of all outgoing neighbors of location i, d_{ij} is the deterministic travel time from location i to location j, $\hat{d}_{jt}$ is a deterministic estimate of the travel time from location j to location t, and $\hat{d}_{it}$ is a deterministic estimate of the travel time from location i to location t. The decision of which outgoing neighbor j to select at location i is made by:

$$j^* \in \operatorname*{argmin}_{j \in N^+(i)} (d_{ij} + \hat{d}_{jt})$$

where j^* is a neighbor that corresponds to the minimum $\hat{d}_{it}$.

For each neighbor j of location i, the Bellman-Ford routing algorithm calculates $\hat{d}_{it}$ using the travel time from location i to location j, and the estimated travel time

from location j to location t. There is no explicit concept of a path when j^* is selected; all that is known is that the path the user will travel on will include j^*. Note that the Bellman-Ford routing algorithm operates on deterministic values. The use of estimates is required because, depending on the implementation, it may take some time before the actual value, d_{jt}, is known.

The Bellman-Ford routing algorithm is useful because it is simple and intuitive. We extend this idea to develop an approach to routing in probabilistic networks. Our intention is to use the framework of the deterministic Bellman-Ford routing algorithm to write:

$$\hat{D}_{it} = \underset{j \in N^+(i)}{\mathcal{MIN}} (X_{ij} + \hat{D}_{jt}) \tag{1.1}$$

where X_{ij} is the arc travel time random variable from location i to location j, and $\hat{D}_{jt}$ is an estimate of the actual travel time random variable from location j to location t. We denote the actual travel time distribution from location j to location t as D_{jt}. The result of Equation 1.1, $\hat{D}_{it}$, is a random variable. By itself, $\hat{D}_{it}$ is of little direct use for routing since we eventually need to make a decision, and this necessarily involves comparing deterministic values. To address this, we instead consider the decision:

$$j^* \in \underset{j \in N^+(i)}{\operatorname{argmin}} (\Gamma(X_{ij}, \hat{D}_{jt})) \tag{1.2}$$

where Γ is a real-valued *operator* that takes as arguments, X_{ij} and $\hat{D}_{jt}$. When we say Γ is an operator, we mean that Γ takes random variables as inputs and returns a single real-value.

Intuitively, we can think of Γ as an operator that specifies the routing objective as a function of the random travel time to neighbor j, and the random travel time from j to t. The deterministic Bellman-Ford routing algorithm can be obtained as a special case:

$$\Gamma(X_{ij}, \hat{D}_{jt}) = X_{ij} + \hat{D}_{jt} = d_{ij} + \hat{d}_{jt}$$

The benefit of this approach is that it allows us to simplify the procedure of making routing decisions with random travel times. In particular, for each neighbor j, it allows us to decompose Equation 1.2 into two parts: X_{ij} and $\hat{D}_{jt}$. We elaborate on this approach in Section 2.7.

1.6 Thesis Contributions and Organization

The contributions of this thesis are:

Survey of Routing We provide a survey of the different approaches to routing in probabilistic networks. We consider routing applications from both data and transportation networks.

Travel Time Distributions We present prior work concerned with the calculation of travel time distributions in probabilistic networks. The majority of this work is from project management literature. We adapt these methods so they can be used in our approach to routing. To the best of our knowledge, this is the first work in routing that is explicitly concerned with calculating travel time distributions to improve the quality of routing.

Routing in Probabilistic Networks We develop an approach to routing in probabilistic networks that uses minimum travel time distributions to make routing decisions. The fundamental idea is that if we know the travel time distributions of the routing options, we can efficiently calculate many different routing objectives. Additional benefits include the ability to specify routing objectives on a per-user basis and the ability to modify routing objectives *en route*.

Numerical Functions In our approach to routing, two functions of random variables need to be calculated repeatedly: the sum and the minimum of random variables. In addition to discussing the efficient implementation of these functions, we also present a data structure for the representation of a random variable.

Practical Implementation We make an effort to keep our ideas suitable for practical implementations. We address and list the additional requirements necessary for an on-line implementation.

The rest of this thesis is organized as follows. Chapter 2 formally defines the notation, assumptions, and our routing procedure. In Chapter 3, we review the relevant research literature. Chapter 3 considers numerous technical results from computer science, operations research, and project management literature. Chapter 4 presents algorithms for calculating the minimum travel time distributions. In Chapter 5, we describe several algorithms that are necessary for the practical use of our routing procedure. Additionally, the numerical implementations of the sum and the minimum of random variables are described. We illustrate our approach to routing on two small networks in Chapter 6. Results on a larger network are also provided to show the performance of the routing procedure in a more practical setting. We also present the performance of the numerical routines for the sum and the minimum of random variables. In Chapter 7, we provide a summary of the issues, our results, and recommendations for further work. We also discuss ideas relevant for on-line implementations.

It is our hope that this work will stimulate further research on routing with random travel times and the calculation of travel time distributions.

[This page intentionally left blank.]

Chapter 2

Preliminaries

This chapter covers preliminary work for the development of this thesis. These preliminaries include our notation, assumptions, and an overview of our routing procedure. In order to understand the difficulty of this problem, we also consider the complexity of routing in probabilistic networks.

2.1 Notation

Let $G = (N, A)$ be a directed graph where N is the set of nodes and $A \subseteq N \times N$ is the set of arcs. Let $m = |A|$ and $n = |N|$ where $|\cdot|$ is the set cardinality operation. The term *digraph* is an abbreviation to denote a directed graph. In this thesis, the term graph refers to a digraph. A graph is also called a *network*.

(i, j) denotes an arc oriented from node i to node j. i and j are called the tail and head nodes, respectively, of arc (i, j). Arc (j, i) is distinct from arc (i, j). An arc of the form (i, i) is called a *self-arc* or *loop*. If, for a pair of nodes i and j in a digraph, there exists multiple arcs from i to j, then we say that G is a *multidigraph*. If there are multiple arcs from node i to node j, the kth arc from i to j is denoted by $(i, j)^k$. If there is only one arc from node i to node j, it is denoted $(i, j)^1$ or as (i, j) with the superscript removed.

If $(i, j) \in A$, nodes i and j are said to be *adjacent*. If $(i, j) \in A$, we also say j is *adjacent* to i. Arc (i, j) is said to be *incident* with nodes i and j. The *outgoing arc adjacency list* for node i is denoted $A^+(i)$ and is defined as the set of arcs (i, j) where j is adjacent to i. The *incoming arc adjacency list* for node i is denoted $A^-(i)$ and is defined as the set of arcs (l, i) where i is adjacent to l. The *outgoing node adjacency list* for node i is denoted $N^+(i)$ and is defined as the set of nodes j where j is adjacent to i. The *incoming node adjacency list* for node i is denoted $N^-(i)$ and is defined as the set of nodes l where i is adjacent to l. In multidigraphs, the size of a node adjacency list need not be the same size as the corresponding arc adjacency list. That is, an arc adjacency list may contain multiple arcs of the form (i, j), while the corresponding node adjacency list contains only distinct nodes j. For notational convenience, let $A(i, j)$ denote the *parallel arc adjacency list*. $A(i, j)$ is a list of all arcs from node i to node j. In other words, $|A^+(i)| = \sum_{j \in N^+(i)} |A(i, j)|$.

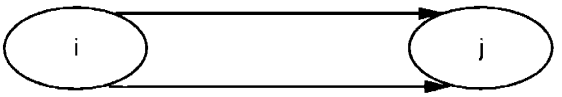


Figure 2-1: Arcs in Parallel.

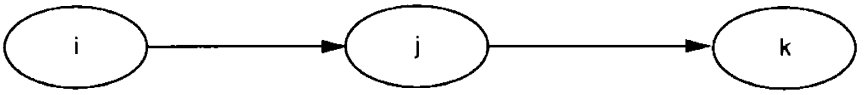


Figure 2-2: Arcs in Series.

Arcs (i,j) and (l,k) are said to be in *parallel* if $i = l$ and $j = k$. In this case, we also say arcs (i,j) and (l,k) form a *parallel component*. Arcs in parallel are displayed in Figure 2-1. Arcs (i,j) and (l,k) are said to be in *series* if $j = l, i \neq k$, and $|A^-(j)| = 1, |A^+(j)| = 1$. In this case, we also say arcs (i,j) and (l,k) form a *series component*. Two arcs in series are displayed in Figure 2-2. A *series reduction* of two arcs $(i,j),(j,k)$ in series replaces $(i,j),(j,k)$ with a single arc from node i to node k. Node j is also removed. A series reduction on two arcs in series decreases m and n by one. A *parallel reduction* on two parallel arcs $(i,j)^1,(i,j)^2$ replaces $(i,j)^1,(i,j)^2$ with a single arc from node i to node j. A parallel reduction on two arcs in parallel decreases m by one. These definitions extend naturally to multiple arcs in series and in parallel. If a graph G can be reduced to a single arc (i,j) via series and parallel reductions, the graph is said to be *series-parallel reducible*. When a graph is series-parallel reducible, then it is called a *series-parallel* graph. For the purpose of this thesis, this definition implies that a series-parallel graph must be acyclic. The removal of any arc (i,j) or any node i from a graph is called a *graph reduction*.

A node s with $|N^-(s)| = 0$ is called a *source* or *origin*. A node t with $|N^+(t)| = 0$ is called a *sink* or *destination*. Source and sink nodes are also called *terminal* nodes.

A *directed path* or *dipath* from source s to sink t is a sequence of arcs (s,i)-(i,j)-(j,k)-...-(h,l)-(l,t) with the following properties:

- The directed path begins with an arc (s,i) and ends with an arc (l,t).
- With the exception of s and t, each head node appears once as a tail node and each tail node appears once as a head node.
- For any arc $(i,j), j \neq t$ in a directed path, there must also be an arc (j,k) in the path.

In a digraph without any parallel arcs, we often denote a path (s,i)-(i,j)-(j,k)-...-(h,l)-(l,t) as the corresponding sequence of nodes, *s-i-j-k-...-h-l-t*. Note that a sequence of nodes is not sufficient to define a dipath in a multidigraph. For example, in Figure 2-1, if we specify a dipath as the nodes *i-j*, we do not know whether the path refers to arc $(i,j)^1$ or arc $(i,j)^2$. We denote the kth path from source s to sink t as π_{st}^k. π_{st} refers to π_{st}^1. Let Π_{st} be the set of all paths from source s to sink t. We also refer to any path π_{st}^k as a (s,t)-path. A *directed cycle* or *dicycle* is a path π_{st}

together with an arc (t, s). In this thesis, the term *path* refers to a directed path, and the term *cycle* refers to a directed cycle.

For a path that contains arcs (l, i) and (i, j), (l, i) is called a *predecessor arc* to (i, j). Similarly, the arc (i, j) is called a *successor arc* to (l, i). Furthermore, for a path that includes (i, j), i is called a *predecessor node* to j, and j is called a *successor node* to i. Two nodes, s and t, are said to be *connected* if there exists a path π_{st} or a path π_{ts} (or both). A graph is said to be connected if every pair of nodes is connected. A graph is said to be *strongly connected* if, for any two nodes s and t, there exists a path π_{st} and a path π_{ts}.

A graph without cycles is called *acyclic*. A digraph (or multidigraph) with no directed cycles is called a *directed acyclic graph* and is denoted $\mathcal{DAG}$. A $\mathcal{DAG}$ with a single source-sink pair is called a *two-terminal* digraph and is denoted $\mathcal{TT}$-$\mathcal{DAG}$.

A comprehensive list of the notation in this thesis is presented in Table 2.1. Several terms are formally defined later.

Table 2.1: Notation.

G	A graph (network) composed of sets N and A. $G = (N, A)$.
N	The set of nodes in G.
A	The set of arcs in G. $A \subseteq N$ x N.
m	The number of arcs in G. $m = \|A\|$.
n	The number of nodes in G. $n = \|N\|$.
G'	G' is a subgraph of G. $G' = (N', A')$.
A'	A' is a subset of the arcs in G. $A' \subseteq A$.
N'	N' is a subset of the nodes in G. $N' \subseteq N$.
$(i, j)^k$	The kth directed arc from i to j. $(i, j)^k \in A$. $i, j \in N$.
(i, j)	$(i, j)^k \in A$, $k = 1$.
s	s is a source (origin) node. $s \in N$.
t	t is a sink (destination) node. $t \in N$.
$\mathcal{DAG}$	Abbreviation for a directed acyclic graph.
$\mathcal{TT}$-$\mathcal{DAG}$	Abbreviation for a directed acyclic graph with two terminals.
Π_{st}	The set of all paths from s to t. $s, t \in N$.
π_{st}^k	The kth path from s to t. $s, t \in N$.
π_{st}	π_{st}^k, $k = 1$.
$A^+(i)$	The outgoing arc adjacency list. $\{(i, j)^k \mid (i, j)^k \in A, j \in N\}$.
$A^-(i)$	The incoming arc adjacency list. $\{(j, i)^k \mid (j, i)^k \in A, j \in N\}$.
$A(i, j)$	The parallel arc adjacency list. $\{(i, j)^k \mid (i, j)^k \in A\}$.
$N^+(i)$	The outgoing node adjacency list. $\{j \mid j \in N, (i, j) \in A\}$.
$N^-(i)$	The incoming node adjacency list. $\{j \mid j \in N, (j, i) \in A\}$.
d_{it}	A deterministic travel time from i to t. $i, t \in N$.
$\hat{d}_{it}$	An estimate of d_{it}.
X_{ij}^k	The arc travel time random variable for $(i, j)^k \in A$.
X_{ij}	X_{ij}^k, $k = 1$.
x_{ij}^k	A realization of the travel time for $(i, j)^k \in A$.
x_{ij}	x_{ij}^k, $k = 1$.

Table 2.1: Notation (continued).

D_{it}	The actual travel time distribution from i to t. $i, t \in N$.
$\hat{D}_{it}$	An estimate of D_{it}.
P_{st}^k	The travel time distribution of π_{st}^k. $s, t \in N$.
P_{st}	P_{st}^k. $k = 1$.
P_{st}^*	The minimum travel time distribution from s to t. $s.t \in N$.
$f_Z(z)$	The probability density function of random variable Z.
$F_Z(z)$	The cumulative distribution function of random variable Z.
$E[Z]$	The expectation of the random variable Z.
$Var(Z)$	The variance of the random variable Z.
$\mathcal{MIN}$	The minimum function of random variables.
$\mathcal{N}(\mu, \sigma^2)$	A normal distribution with mean μ and variance σ^2.
$\mathcal{U}(a, b)$	A uniform distribution from a to b.
$\gamma(\alpha, \beta)$	A gamma distribution with parameters α and β.
$\mathcal{EXP}(\frac{1}{\lambda})$	An exponential distribution with mean $\frac{1}{\lambda}$.
$\cup$	The set union operator.
$\cap$	The set intersection operator.
$\setminus$	The set difference operator.

2.2 Network Model

With this notation, we now generalize the examples in Chapter 1 to formalize our model. We consider a network $G = (N, A)$ with the following properties:

- G is strongly connected.
- G does not contain self-arcs.
- Associated with each arc (i, j) is a positive random variable X_{ij} which is the travel time along arc (i, j).

In this thesis, we refer to a network that satisfies these properties as a *probabilistic* network. Probabilistic networks are also known as *stochastic* networks. Note that this model is quite general and can be used for a variety of applications.

2.3 Assumptions

In addition to the network model presented in Section 2.2, we make several additional assumptions:

2.3.1 Arc Travel Time Independence

We assume the arc travel times in G are independent. That is,

$$f_{X_{ij}, X_{kl}, \ldots}(x_{ij}, x_{kl}, \ldots) = f_{X_{ij}}(x_{ij}) f_{X_{kl}}(x_{kl}) \ldots$$

for all arcs $(i,j),(k,l),\ldots \in A$. Intuitively, this means that knowing the travel time on any arc (i,j) does not provide us with any information of the travel time on any other arc (k,l). Without this assumption, calculating the minimum travel time distribution is much more difficult. This difficulty arises because the analysis of the sum and the minimum of random variables is much more complicated with dependent arc travel times. Such an analysis requires knowledge of the arc travel time joint distribution functions, and careful random variable conditioning. This calculation is possible by hand when the network is simple and the distributions are well-known and easy to work with. However, in the context we envision our algorithms to be operating in, we cannot assume anything about the complexity of the network or the travel time distributions.

Note that this assumption does not mean the *paths* within a network are independent. Indeed, any two paths that share a common arc (i,j) are *not* independent from each other. This dependence represents one of the core difficulties in working with probabilistic networks.

It may be argued that arcs in numerous application networks are dependent. In particular, it is not hard to imagine a transportation network where knowledge of the travel time on one arc suggests a travel time or range of travel times on another arc. However, we make the independence assumption for two reasons:

- The computational requirements can be overwhelming for networks with dependent arcs. Without independent arcs, it is much more difficult to analyze the sum and the minimum of random variables.
- There is limited prior research that attempts to make routing decisions based on the explicit calculation of travel time distributions. As such, we necessarily begin this area of research under a simplified scenario.

2.3.2 Arc Travel Time Distributions

The assumptions on the arc travel time probability density functions, $f_{X_{ij}}(x_{ij})$, are as follows:

- $f_{X_{ij}}(x_{ij})$ is defined over $(0,\infty)$ where $f_{X_{ij}}(x_{ij}) \geq 0 \; \forall x_{ij} \in [a,b]$ for some $a,b \in \mathbb{R}^+, a < b$ and 0 otherwise. For the purpose of implementation, we consider $f_{X_{ij}}(x_{ij}) = 0$ if $f_{X_{ij}}(x_{ij}) \leq \epsilon$ for some sufficiently small ϵ (i.e. $\epsilon \approx 0$). For example, if $X_{ij} = \mathcal{N}(10,8)$ is defined over $[0.1, 19.9]$, then we consider $f_{X_{ij}}(x_{ij})$ valid since $f_{X_{ij}}(x_{ij}) < \epsilon = 0.0003 \; \forall x_{ij} < 0.1$ (and $f_{X_{ij}}(x_{ij}) < \epsilon = 0.0003 \; \forall x_{ij} > 19.9$). The finite range $[a,b]$ is only necessary for implementation.
- $f_{X_{ij}}(x_{ij})$ is continuous over $[a,b]$. This assumption adds realism in the sense that given any two travel time realizations $x'_{ij} < x''_{ij}$, we assume there exists another realization x'''_{ij} such that $x'_{ij} < x'''_{ij} < x''_{ij}$.
- $\lim_{x_{ij} \to a^+} f_{X_{ij}}(x_{ij}) = 0$ and $\lim_{x_{ij} \to b^-} f_{X_{ij}}(x_{ij}) = 0$. This is necessary to prevent discontinuities from being introduced in the minimum travel time distributions calculated by the algorithms in Chapter 4.

- The probability density functions that we work with are smooth (C^1 functions, 1-smooth). Strictly speaking, this is not a formal requirement, but like the assumption of continuity, this adds realism in the sense that we would generally expect travel time distributions to be smooth.

In Chapter 5, we discuss how we represent probability density functions. Interpolation is essential to this representation. The most important consequence of the these assumptions is that, under these assumptions, our use of interpolation produces accurate results. In the context of the work in Chapter 5, Appendix A discusses the problem that can arise if these assumptions are not met.

Note that, aside from these conditions, we do not assume any particular distribution or distribution characteristics.

2.3.3 Cycles

We assume that traversing a cycle is never beneficial. This assumption is obvious in deterministic networks, but is much more subtle in probabilistic networks. Consider the graph in Figure 2-3. Without loss of generality, assume X_{sj} is a constant 1, X_{ji} and X_{ij} have identical distributions $\mathcal{U}(1,2)$, and X_{jt} has distribution $\mathcal{U}(3,7)$. Consider the question of whether the inequality $x_{sj} + x_{ji} + x_{ij} + x_{jt} > x_{sj} + x_{jt}$ is always true. It will always be true if the "left-hand-side" realizations of x_{sj}, x_{jt} are the same as the "right-hand-side" realizations of x_{sj}, x_{jt}. However, ensuring that these realizations are the same is tricky. In practice, there might be a time component to the realizations. For example, a user arriving at node j at two different times might be given two different realizations for x_{jt}. If this is the case, then there may be some probability that traversing the cycle will result in a faster travel time from node s to node t (particularly if x_{jt} is large the first time it is realized). In a simulation, a random number generator may add a time component to the realizations. In any case, we take the point of view that in practical applications, we intuitively would not traverse a cycle. Note that this assumption only applies to our routing decisions; we make no assumptions as to whether the underlying network G contains (or does not contain) cycles.

2.3.4 Congestion

In a network that is subject to *congestion*, the number of users of a particular arc (i,j) affects the travel time along (i,j). We assume that the networks in this thesis are not subject to congestion.

2.4 Two Fundamental Functions of Random Variables

The sum and the minimum of random variables are two functions of random variables that are used extensively in this thesis. We review these functions to provide

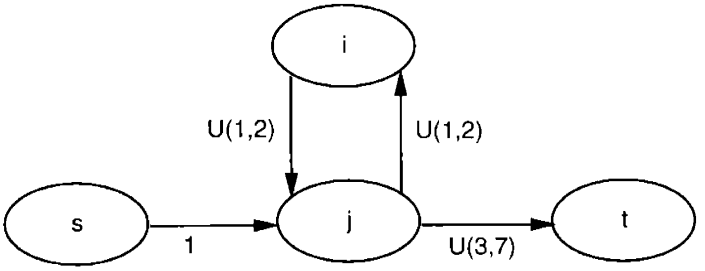


Figure 2-3: Cycles in probabilistic networks. We take the point of view that in practical applications, we intuitively would not traverse a cycle. This assumption is obvious in deterministic networks, but is much more subtle in probabilistic networks. See Section 2.3.3.

the foundation for the implementation of the corresponding numerical routines (see Section 5.2).

2.4.1 The Sum of Random Variables

The random variable, $C = A + B$, is the sum of the random variables A and B, and can be obtained using convolution. The cumulative distribution of C is derived via the following steps:

$$\begin{aligned} F_C(c) &= \Pr(C \leq c) \\ &= \Pr(A + B \leq c) \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{c-a} f_{A,B}(a, b)\, db\, da \end{aligned}$$

If A and B are independent, we further calculate:

$$\begin{aligned} F_C(c) &= \int_{-\infty}^{\infty} \int_{-\infty}^{c-a} f_A(a) f_B(b)\, db\, da \\ &= \int_{-\infty}^{\infty} f_A(a) F_B(c-a)\, da \end{aligned}$$

With A and B independent, the density of C, $f_C(c)$, is given by:

$$\begin{aligned} f_C(c) &= \frac{dF_C(c)}{dc} \\ &= \int_{-\infty}^{\infty} f_A(a) \frac{dF_B(c-a)}{dc}\, da \\ &= \int_{-\infty}^{\infty} f_A(a) f_B(c-a)\, da \end{aligned} \tag{2.1}$$

This derivation is a standard part of most probability textbooks (see [6] and [31]). Equation 2.1 is the convolution of the two probability density functions $f_A(a)$ and

$f_B(b)$, and is denoted $f_A(a) * f_B(b)$. The sum of independent random variables is associative and commutative.

Let P_{st}^k be the random variable of travel time along the kth path, π_{st}^k, from node s to node t. Then P_{st}^k is defined as:

$$P_{st}^k = \sum_{(i,j)\in\pi_{st}^k} X_{ij}$$

2.4.2 The Minimum of Random Variables

The random variable, $C = \mathcal{MIN}(A, B)$, is the minimum of the random variables A and B. $\mathcal{MIN}$ is capitalized to denote that C is a random variable. The $\mathcal{MIN}$ function is derived in both [24] and [6]. For completeness, we include a derivation that follows [31]. The cumulative joint distribution of A and B is denoted $F_{A,B}(c, c) = \Pr(A \le c, B \le c)$. The cumulative distribution of $C = \mathcal{MIN}(A, B)$ is given by:

$$\begin{aligned}
C &= \mathcal{MIN}(A, B) \\
F_C(c) &= \Pr(C \le c) \\
&= \Pr(\mathcal{MIN}(A, B) \le c) \\
&= \Pr(B \le c, A > B) \cup \Pr(A \le c, A \le B) \\
&= \Pr(B \le c, A > B) + \Pr(A \le c, A \le B) \\
&= 1 - \Pr(A > c, B > c) \\
&= 1 - \Pr(\infty > A > c, \infty > B > c) \\
&= 1 - (F_{A,B}(\infty, \infty) - F_{A,B}(c, \infty) - F_{A,B}(\infty, c) + F_{A,B}(c, c)) \\
&= 1 - 1 + F_A(c) + F_B(c) - F_{A,B}(c, c) \\
&= F_A(c) + F_B(c) - F_{A,B}(c, c).
\end{aligned}$$

If A and B are independent, we further calculate:

$$\begin{aligned}
F_C(c) &= F_A(c) + F_B(c) - F_{A,B}(c, c) \\
&= F_A(c) + F_B(c) - F_A(c)F_B(c).
\end{aligned}$$

With A and B independent, the density of C, $f_C(c)$, is:

$$\begin{aligned}
f_C(c) &= \frac{dF_C(c)}{dc} \\
&= \frac{d(F_A(c) + F_B(c) - F_{A,B}(c, c))}{dc} \\
&= \frac{d(F_A(c) + F_B(c) - F_A(c)F_B(c))}{dc} \\
&= f_A(c) + f_B(c) - f_A(c)F_B(c) - f_B(c)F_A(c) \\
&= f_A(c)[1 - F_B(c)] + f_B(c)[1 - F_A(c)]
\end{aligned} \tag{2.2}$$

Like the sum of random variables, the minimum of random variables is associative

and commutative. Furthermore, if $C = \mathcal{MIN}(A + x, B + x)$ for $x \in \mathbb{R}$, then $C = \mathcal{MIN}(A, B) + x$ since:

$$\begin{aligned}
C &= \mathcal{MIN}(A + x, B + x) \\
F_C(c) &= \Pr(\mathcal{MIN}(A + x, B + x) \leq c) \\
&= \Pr(B + x \leq c, A + x > B + x) \cup \Pr(A + x \leq c, A + x \leq B + x) \\
&= \Pr(B \leq c - x, A > B) \cup \Pr(A \leq c - x, A \leq B) \\
&= \Pr(\mathcal{MIN}(A, B) \leq c - x) \\
&= \Pr(\mathcal{MIN}(A, B) + x \leq c) \\
C &= \mathcal{MIN}(A, B) + x.
\end{aligned}$$

2.5 The Minimum Travel Time Distribution

Let P^*_{st} denote the minimum travel time distribution from node s to node t. One way to think about P^*_{st} is to repeatedly execute the following steps:

1. Fix each arc $(i, j) \in A$ to a random realization x_{ij}.
2. Run a one-to-one fastest path algorithm from node s to node t.
3. Record the travel time of the fastest path to node t.

After enough iterations of these steps, the distribution of the fastest travel times from node s to node t will emerge. This is P^*_{st}.

P^*_{st} can be expressed using the $\mathcal{MIN}$ function and $|\Pi_{st}|$, the number of paths from node s to node t. Formally, P^*_{st} is given by:

$$P^*_{st} = \mathcal{MIN}(P^1_{st}, P^2_{st}, \ldots, P^{|\Pi_{st}|}_{st})$$

As outlined in [24], another way to think of P^*_{st} is presented in Algorithm 2.1. The two main functions in Algorithm 2.1 are the sum and the minimum of random variables. The order of the arguments to these functions does not matter because these functions are commutative. Similarly, P^*_{st} can be computed iteratively because the sum and the minimum functions are associative.

While it is possible to calculate P^*_{st} in this manner, there are two problems with this approach:

- Algorithm 2.1 requires enumerating all paths $\pi^k_{st} \in \Pi_{st}$, which is not efficient as the number of arcs and nodes increases (G is not assumed to be acyclic).
- If any paths share arcs, then the paths are dependent. The minimum function is much more difficult to evaluate when the random variables are dependent. Such a calculation would require knowledge of joint distributions.

Input: $G = (N, A)$. $s, t \in N$. $X = \{X_{ij} | (i,j) \in A\}$.
Output: P^*_{st}.
Step 0: Initialization.
$P^1_{st} \leftarrow \sum_{(i,j)\in\pi^1_{st}} X_{ij}$
$P^*_{st} \leftarrow P^1_{st}$
Step 1: Calculate P^k_{st} for each path and update the minimum travel time distribution.
for all $\pi^k_{st} \in \Pi_{st} \setminus \pi^1_{st}$ **do**
$P^k_{st} = \sum_{(i,j)\in\pi^k_{st}} X_{ij}$
$P^*_{st} = \mathcal{MIN}(P^*_{st}, P^k_{st})$

Algorithm 2.1: Calculation of the Minimum Travel Time Distribution.

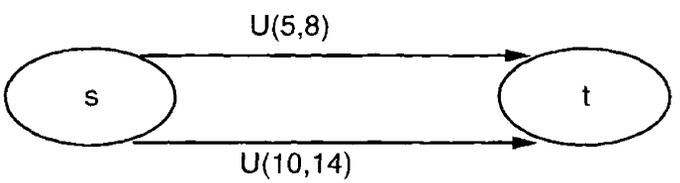


Figure 2-4: Two Travel Time Distributions from Node s to Node t.

Note that the actual travel time from node s to node t may not be drawn from P^*_{st}. To emphasize this point, consider Figure 2-4. There are two paths from node s to node t. One path has an actual travel time distribution of $\mathcal{U}(5,8)$ and the other path has an actual travel time distribution of $\mathcal{U}(10,14)$ so that $P^*_{st} = \mathcal{MIN}(\mathcal{U}(5,8), \mathcal{U}(10,14))$. Observe that $\mathcal{U}(10,14)$ is *dominated* by $\mathcal{U}(5,8)$ in the sense that the highest possible realization for $\mathcal{U}(5,8)$ will always be less than the lowest possible realization for $\mathcal{U}(10,14)$. This means $P^*_{st} = \mathcal{U}(5,8)$. However, $D_{st} \neq P^*_{st}$ since D_{st} depends on which path is selected, which, in turn, depends on the routing objective. In other words, if the user selects the path with travel time $\mathcal{U}(5,8)$ then $D_{st} = \mathcal{U}(5,8)$ and, similarly, if the user selects the path with $\mathcal{U}(10,14)$ then $D_{st} = \mathcal{U}(10,14)$. However, regardless of what the routing objective is, P^*_{st} always equals $\mathcal{U}(5,8)$.

2.6 The Complexity of Calculating Travel Time Distributions

It turns out that calculating the minimum travel time distribution is a difficult problem. We define this problem as:

Definition (Minimum travel time distribution calculation). *Given a digraph $G = (N, A)$, $s, t \in N$, and an independent, positive travel time random variable X_{ij} associated with each $(i,j) \in A$, find the minimum travel time distribution from s to t.*

In [45], Valiant shows that a related problem known as **s $\rightarrow$ t connectedness**,

is $\mathcal{NP}$-Hard[1]. [35] and [34] extend and complement Valiant's work. Formally, the **s → t connectedness** problem is defined as:

Definition (s → t connectedness). *Given a graph $G = (N, A)$ and $s, t \in N$, find the number of subgraphs of G where there is a path from s to t.*

Lemma 1. *The* **s → t connectedness** *problem applies to both undirected and directed graphs and is $\mathcal{NP}$-Hard.*

Proof. See [45] □

Valiant observes **s → t connectedness** is related to the **s → t connectedness reliability** problem. The **s → t connectedness reliability** problem is defined as:

Definition (s → t connectedness reliability). *Given a digraph $G = (N, A)$ and $s, t \in N$, and the probability p of each arc (i, j) failing independently of other arcs, find the probability there is at least one directed path from s to t.*

Lemma 2. *The* **s → t connectedness reliability** *problem applies to both undirected and directed graphs and is $\mathcal{NP}$-Hard.*

Proof. The essence of the proof is that in order to calculate the probability of a directed path from s to t, we need to count the subgraphs where there exists a directed path from s to t. This means that we need to solve the **s → t connectedness** problem first which is $\mathcal{NP}$-Hard. For the complete proof, see [45] for the foundation and [35], [34] for an extended analysis. □

The **s → t connectedness reliability** problem provides the necessary insight to understand the difficulty of calculating the minimum travel time distribution. To see this, consider a modified version of the **s → t connectedness reliability** problem. In this modified version, rather than each arc (i, j) failing with probability p, the travel time along (i, j) is set to ∞ with probability p, and $\eta \in \mathbb{R}^+$ otherwise. This gives rise to the following problem:

Definition (s → t finite-connectedness). *Given a digraph $G = (N, A)$, $s, t \in N$, and the independent and identical probability p of each arc (i, j) taking travel time ∞ (and $\eta \in \mathbb{R}^+$ otherwise), find the probability that there is at least one directed path from s to t with finite length.*

Lemma 3. *The* **s → t finite-connectedness** *problem is $\mathcal{NP}$-Hard.*

Proof. This follows directly from Lemma 2. Finding a path from s to t with finite length is equivalent to finding the existence of a path from s to t in the **s → t connectedness reliability** problem. □

Now we observe that the **s → t finite-connectedness** problem is the same as calculating the minimum travel time distribution when the travel times on $(i, j) \in A$ are identical Bernoulli random variables. This leads to the conclusion:

[1]The reference to Valiant's work was made in [41] but an explanation was not provided. We provide our own explanation for clarity.

Theorem 1. *The problem of calculating the minimum travel time distribution is $\mathcal{NP}$-Hard.*

Proof. As noted, in the case where arc travel times are identical Bernoulli random variables, calculating the minimum travel time distribution is equivalent to solving the **s → t finite-connectedness** problem, which is $\mathcal{NP}$-Hard. Furthermore, since this problem is $\mathcal{NP}$-Hard when the arc travel times are identical Bernoulli random variables, then it must also be $\mathcal{NP}$-Hard for more general arc travel times distributions.

An alternative view is to modify the **s → t finite-connectedness** problem to *find the probability that there is at least one directed path from s to t with finite length less than or equal to* $\mu \in \mathbb{R}^+$. This modified **s → t finite-connectedness** problem can also be used to calculate the minimum travel time distribution. However, to do this, we would need solve the modified **s → t finite-connectedness** problem for multiple values of μ, which is clearly $\mathcal{NP}$-Hard. □

2.7 Routing in Probabilistic Networks

In Chapter 1, we present an approach to routing in probabilistic networks that is based on the ideas of the Bellman-Ford routing algorithm. In particular, we define Equation 1.2 as:

$$j^* \in \underset{j \in N^+(i)}{\operatorname{argmin}} \left(\Gamma(X_{ij}, \hat{D}_{jt})\right)$$

In order to implement this, we further decompose Γ to:

$$j^* \in \underset{j \in N^+(i)}{\operatorname{argmin}} \left(\Gamma(\Phi(X_{ij}), \Psi(\hat{D}_{jt}))\right) \tag{2.3}$$

where $\Phi(X_{ij})$ is some metric on the random variable X_{ij}, and $\Psi(\hat{D}_{jt})$ is some metric on the random variable $\hat{D}_{jt}$. Unlike Γ, which returns a real-value, Φ and Ψ are meant to be abstractly defined; they need not return real-values and could return random variables. Note that we have still included $\hat{D}_{jt}$, which is an estimate of the actual travel time distribution from node j to node t. In Section 1.4, we observe that it is only possible to calculate the actual travel time distribution if a routing objective is specified *a priori*. Since one of our primary motivations is to avoid the specification of a routing objective *a priori*, we instead approximate $\hat{D}_{jt}$ with the minimum travel time distribution, P^*_{jt}. In some sense, P^*_{jt} "encapsulates" the fastest travel times over all the paths from node j to node t.

In Table 2.2, several possible decompositions of Γ are provided. For example, in the first objective,

$$\Gamma(\Phi(X_{ij}), \Psi(P^*_{jt})) = \Phi(X_{ij}) + \Psi(P^*_{jt}) = E[X_{ij}] + E[P^*_{jt}]$$

In other words, Γ represents the expected travel time from node i to node t via node j.

Obj.	$\Gamma(\Phi(X_{ij}), \Psi(P^*_{jt}))$	$\Phi(X_{ij})$	$\Psi(P^*_{jt})$
1	$\Phi(X_{ij}) + \Psi(P^*_{jt})$	$E[X_{ij}]$	$E[P^*_{jt}]$
2	$\Psi(P^*_{jt})$		$E[P^*_{jt}]$
3	$\Phi(X_{ij}) + \Psi(P^*_{jt})$	$E[X_{ij}] + \theta \cdot Var(X_{ij})$	$E[P^*_{jt}] + \theta \cdot Var(P^*_{jt})$
4	$\Phi(X_{ij}) + \Psi(P^*_{jt})$	x_{ij}	$E[P^*_{jt}] + \theta \cdot Var(P^*_{jt})$
5	$-\Psi(P^*_{jt})$		$\Pr(P^*_{jt} \leq \xi)$
6	$-\Pr(\Phi(X_{ij}) + \Psi(P^*_{jt}) \leq \xi)$	X_{ij}	P^*_{jt}

Table 2.2: Examples of Routing Objectives.

To solve Equation 2.3, we would need to calculate $E[X_{ij}] + E[P^*_{jt}]$ for each $j \in N^+(i)$. j^* is then selected such that $E[X_{ij}] + E[P^*_{jt}]$ is the minimum of $E[X_{ij}] + E[P^*_{jt}]$ over all $j \in N^+(i)$. The second objective disregards the initial arc (i, j) and is only concerned with $E[P^*_{jt}]$. This might occur in the case where the possible realizations of X_{ij} have little effect on the total travel time. The third objective attempts to strike a balance between expectation and variance. The user-specified parameter $\theta \in \mathbb{R}^+ \cup \{0\}$ is meant to reflect the importance of variance to the user. If a user values low variance in travel time, the user should choose a large value of θ to reflect the relative disutility of high variance in travel time. In the fourth objective, $\Phi(X_{ij})$ is set to a current realization x_{ij} of the travel time along arc (i, j). Such a realization might be obtained by an information system. This objective attempts to factor on-line, updated information into the routing objective. In the fifth objective, $\Psi(P^*_{jt})$ factors in the probability that the travel time from node j to node t will be less than a given value ξ. Note that in the context of Equation 2.3 we take the negative of $\Psi(P^*_{jt})$ since we want to maximize $\Psi(P^*_{jt})$. Intuitively, $\Psi(P^*_{jt})$ represents a certain *confidence* that the travel time will be less than ξ. The sixth objective is similar to the fifth objective except that it includes X_{ij}. Unlike the previous objectives, in the sixth objective Γ cannot be decomposed by calculating $\Phi(X_{ij})$ and $\Psi(P^*_{jt})$ separately. Therefore, $\Phi(X_{ij}) = X_{ij}$ and $\Psi(P^*_{jt}) = P^*_{jt}$ and we want the probability that $X_{ij} + P^*_{jt}$ is less than ξ. In a similar manner to the fifth objective, we take the negative of this probability since we want to maximize this probability.

Clearly, there are many possible decompositions of Γ. Analogous to the Bellman-Ford routing algorithm, we can think of $\Phi(X_{ij})$ as a metric that judges the weight of the travel time of each adjacent arc (i, j). $\Psi(P^*_{jt})$ is a metric that judges the weight of the travel time distribution from node j to node t.

Algorithmically, our routing procedure is presented in Algorithm 2.2. Within this procedure, there are two problems to address:

- Calculating any objective in Table 2.2 is straight-forward *if* we know the minimum travel time distribution P^*_{jt} for each $j \in N^+(i)$. The calculation of P^*_{jt} is the subject of Chapter 4.

Input: $G = (N, A)$. $i, t \in N$. $X = \{X_{ij} | (i,j) \in A\}$.
Output: $j^* \in N^+(i)$.
  Step 0: Initialization.
  $\Gamma^* \leftarrow \infty$
  Step 1: Find j with the minimum Γ.
  for all $j \in N^+(i)$ **do**
    if $\Gamma(\Phi(X_{ij}), \Psi(P^*_{jt})) < \Gamma^*$ **then**
      $j^* \leftarrow j$
      $\Gamma^* \leftarrow \Gamma(\Phi(X_{ij}), \Psi(P^*_{jt}))$

Algorithm 2.2: The Routing Procedure.

Input: $G = (N, A)$. $i, t \in N$. $X = \{X_{ij} | (i,j) \in A\}$
Output: $j^* \in N^+(i)$.
  Step 0: Initialization.
  $\Gamma^* \leftarrow \infty$
  for all $j \in N^+(i)$ **do**
    Step 1: Generate a $\mathcal{TT}$-$\mathcal{DAG}$ G' from G from node j to node t.
    Step 2: Calculate P^*_{jt} on G'.
    Step 3: Find j with the minimum Γ.
    if $\Gamma(\Phi(X_{ij}), \Psi(P^*_{jt})) < \Gamma^*$ **then**
      $j^* \leftarrow j$
      $\Gamma^* \leftarrow \Gamma(\Phi(X_{ij}), \Psi(P^*_{jt}))$

Algorithm 2.3: The Complete Routing Procedure.

- A more subtle problem is that the calculation of P^*_{jt} should be made on a $\mathcal{TT}$-$\mathcal{DAG}$ from node j to node t, rather than on G. In other words, the calculation of P^*_{jt} need only consider parts of G that can *potentially* be used in traveling from node j to node t. The problem of deciding what parts of G are relevant to the calculation of P^*_{jt} is the subject of Chapter 5.

Adding these two steps to Algorithm 2.2 yields the complete Algorithm 2.3.

Chapter 3

Relevant Research

In this chapter, we review research relevant to routing in probabilistic networks. Work on the calculation of distributions in probabilistic networks appears to have originated in the early 1960s with project management research. Because of the difficulty of working with travel time distributions, most research focuses on using the least expected travel time as the routing objective. The ideas in this chapter come from project management, probability, and graph theory. We also touch upon issues related to the practical implementation of our work.

3.1 Introduction

The general problem of routing is a well-researched area and has applications in many fields. There are many variants to the problem and, within each variant, there are many different approaches. Examples of application areas include transportation, logistics, data networking, and project management.

As outlined in Chapter 1, our particular problem is that of routing in probabilistic networks. In researching this problem, we classify the literature into two basic approaches. The first approach focuses on a specific routing objective. For example, if we focus on the *least expected* travel time, we can set X_{ij} to $E[X_{ij}]$ and solve a deterministic shortest path problem. The substitution effectively turns the probabilistic network into a deterministic network. Initially, this approach makes sense both intuitively and computationally. However, it is a simplified approach as it does not take advantage of information that can be derived from travel time random variables. To address this, the second basic approach attempts to calculate P^*_{jt}, the minimum travel time distribution.

3.2 Specific Routing Objectives

Defining the proper routing objective for a probabilistic network is surprisingly difficult. A natural first step is to replace each arc travel time X_{ij} with its expectation $E[X_{ij}]$ and consider routing algorithms for deterministic networks. A comprehensive study of deterministic routing algorithms is presented in [30].

Further complicating the issue is the notion of system-oriented *optimal* routing and user-oriented *shortest path* routing. System-oriented optimal routing attempts to achieve an optimum aggregated across an entire system. User-oriented shortest path routing attempts to find an optimum that is user-specific. Intuitively, user-optimal routing is equivalent to letting each user act in their own best interest. Such behavior is good for an individual user but may not be best for the system as a whole.

There is also the notion of *congestion*, whereby each additional user of a network further degrades the performance of the network. We assume that the networks in this thesis are not subject to congestion.

Additionally, there is the notion of *adaptive* routing where decisions are made *en route* instead of *a priori*. If adaptive routing is used on networks subject to congestion, it may be possible to create oscillating behavior by switching between alternatively uncongested paths. In data networks, there has been work into dampening such oscillations ([4]).

Closely related to adaptive routing is the notion of *on-line* and *off-line* systems. On-line systems are typically used operationally where routing calculations are constantly updated as new information about the network becomes available. In an off-line system, all routing calculations are made with information that is known *a priori*. Consequently, in such a system, the routing decisions are made before any actual routing occurs.

3.3 Routing Objectives in Probabilistic Networks

Research in routing on probabilistic networks tends to consider a single objective such as the least expected travel time. Recent work has started to focus on adaptive objectives that are calculated and revised *en route*. Fu includes a taxonomy of the *Shortest Path Routing Problem* in [15].

Fu discusses an adaptive routing algorithm that attempts to determine the best neighbor $j^* \in N^+(i)$ for each node i that is traversed in a probabilistic network ([15]). Fu assumes that at node i, on-line realizations of the travel times $\{x_{ij} | j \in N^+(i)\}$ are made known. Using these realizations, Fu states the problem of finding the minimum expected travel time from node i to node t recursively as:

$$j^* \in \underset{j \in N^+(i)}{\operatorname{argmin}} (x_{ij} + E[D_{jt}])$$

Fu's approach fits into our approach with $\Gamma(\Phi(X_{ij}), \Psi(\hat{D}_{jt})) = \Phi(X_{ij}) + \Psi(\hat{D}_{jt})$ and $\Phi(X_{ij}) = x_{ij}$ and $\Psi(\hat{D}_{jt}) = E[D_{jt}]$.

Since Fu specifies a routing objective *a priori*, he can calculate D_{jt} recursively. With an appropriate boundary condition this equation forms a dynamic program. Fu points out two difficulties with this approach. First, an acyclic network is required to compute D_{jt} and this is not typical in models of practical networks. This issue was mentioned at the end of Chapter 2. Second, and more critically, D_{jt} needs to be calculated iteratively. The latter problem is addressed by Fu via a "point estimate"

approximation method. This approximation method estimates the moments of a function of random variables from the moments of its random variable arguments. This reduces the complexity of the problem to discrete summations using means and variances. From here, a label-correcting algorithm is proposed with an efficiency equivalent to label-correcting algorithms for the calculation of deterministic shortest paths. Note that Fu does not explicitly calculate D_{jt} but rather estimates $E[D_{jt}]$.

Tsitsiklis and Polychronopoulos find the minimum expected cost with random arc costs [32]. In this paper, it is assumed that once an arc cost realization is made, it is fixed. Two representations of arc costs are discussed. The first represents the costs as a known joint density function and the second represents the costs as independent arc costs. The solution approach is based upon dynamic programming where the stage of the program is equivalent to the amount of deterministic information that is known about the network. The complexity of the problem leads Tsitsiklis and Polychronopoulos to derive approximate solution algorithms.

In [2] and [3], Bertsekas provides a comprehensive analysis of the stochastic shortest path problem using dynamic programming. The objective is to minimize the expected cost when the termination state of the program is reached. Bertsekas provides formal definitions of optimal policies, and discusses the use of the value-iteration and policy-iteration dynamic programming solution methods. Bertsekas comments that these methods may not be suitable for large systems and suggests two alternate approaches: Monte Carlo simulation and neuro-dynamic programming (reinforcement-learning).

Sen et al. [38] formulate and solve a binary program with a quadratic objective. They make the same observations that we make in Chapter 1 and, consequently, they consider variance in their objective. More specifically, for a source i and sink t, they consider a hybrid of expectation and variance:

$$\underset{\pi_{st}^k \in \Pi_{st}}{\operatorname{argmin}} E[P_{st}^k] + \frac{\theta}{2} Var(P_{st}^k)$$

In the program, π_{st}^k is decomposed into binary variables, $z_{ij} \in \{0, 1\}$, that represent whether arc (i, j) is included in the optimal path. The actual solution method involves a relaxation of the binary constraints and the solving of a series of related quadratic programs.

3.4 Travel Time Distributions

The formal concept of the distribution of time in probabilistic networks appears to have been defined by Martin in 1965 [26]. Martin is concerned with the distribution of time in a $\mathcal{TT}$-$\mathcal{DAG}$. More specifically, his work is motivated by the two project management problems, Program Evaluation and Review Technique (PERT) and Critical Path Method (CPM). While the originators of PERT and CPM considered two approaches to randomness in project duration time, neither approach is as general as Martin's approach to calculating the random variable of project duration time. One

of the two original approaches is to replace the project duration time random variables with their expected values, while the other approach requires three estimates of project duration time: one optimistic, one average, and one pessimistic [25],[9].

The formal problem Martin addresses is essentially the same as our problem of calculating P^*_{jt} given nodes j and t. A $\mathcal{TT}$-$\mathcal{DAG}$ is assumed with source s and a sink t. Since Martin is motivated by PERT/CPM networks, each arc (i, j) represents a project *activity*. Associated with each arc (i, j) is an independent random variable X_{ij} of the activity duration time. Martin is concerned with calculating the distribution of the *maximum* project duration time. The activity duration time on each arc is analogous to our notion of arc travel time. Even though Martin focuses on the distribution of maximum project duration time, we can directly apply Martin's ideas to our problem of calculating of the minimum travel time distribution.

For the purpose and clarity of this thesis, we reclassify Martin's work as research on minimum travel time distributions. If we replace every application of the maximum of random variables in Martin's work with the minimum, then the problem of calculating P^*_{jt} is exactly equivalent to Martin's.

3.4.1 Series-Parallel Graphs

One of Martin's observations is that the minimum travel time distribution can be calculated exactly and efficiently on series-parallel graphs. The key point is that for any two arcs in series, an equivalent travel time distribution can be calculated by convolving the density functions of the travel time random variables of the arcs in series. For any two arcs in parallel, the minimum travel time distribution can be calculated by taking the minimum of the travel time random variables of the arcs in parallel.

Given a source s and sink t, Martin describes an efficient algorithm that recursively traverses a series-parallel $\mathcal{TT}$-$\mathcal{DAG}$ G' and reduces it to a single arc (s, t) with $X_{st} = P^*_{st}$. Whenever arcs in series are encountered, they are replaced by a new, single arc whose travel time density function is the result of the convolution of the density functions of the travel times of the arcs in series. Whenever parallel arcs are encountered, they are replaced by a new, single arc whose travel time is the minimum of the arc travel time random variables of the arcs in parallel. Intuitively, it not hard to see that the repeated application of these series and parallel reductions will reduce G' to a two-node, single arc network with $X_{st} = P^*_{st}$.

Relevant work by Duffin [8] and later by Lawler, Tarjan, and Valdes [44] clarifies the conditions for a graph to be series-parallel reducible. An extensive analysis of this problem is provided in [43]. A $\mathcal{DAG}$ subgraph G' of G is *homeomorphic* to a $\mathcal{DAG}$ G if the subgraph G' can be obtained by repeatedly removing arcs and performing series reductions on G. Given this definition, the following result is proved in [44] but is based on [8]:

Theorem 2 (Series-Parallel Reducibility of Graphs). *A digraph G is series-parallel reducible if and only if G is not homeomorphic to the graph in Figure 3-1.*

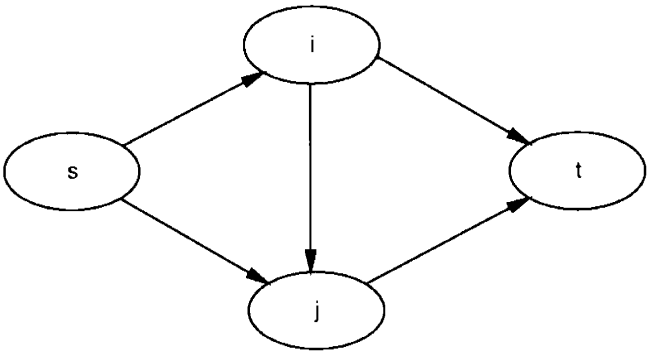


Figure 3-1: The Wheatstone Bridge.

In the literature, the graph in Figure 3-1 has several names including the *Wheatstone Bridge*, the *interdictive graph*, and the *forbidden subgraph*. In this thesis, we refer to Figure 3-1 as the *Wheatstone Bridge*. Intuitively, this theorem states that a graph G is not series-parallel if the Wheatstone Bridge is "embedded" in G. In [44], Lawler, Tarjan, and Valdes present an efficient algorithm to determine if a graph G is series-parallel reducible.

3.4.2 Non-Series-Parallel Graphs

If a graph G is homeomorphic to Figure 3-1, then Theorem 2 tells us G is not series-parallel. In this case, the analysis of the minimum travel time distribution becomes much more difficult. This difficulty is evident in the calculation of the minimum travel time distribution on the Wheatstone Bridge.

We have the following travel times distributions for the three possible paths $\{\pi^1_{st}, \pi^2_{st}, \pi^3_{st}\}$ from node s to node t:

$$\begin{aligned} P^1_{st} &= X_{si} + X_{it} \\ P^2_{st} &= X_{sj} + X_{jt} \\ P^3_{st} &= X_{si} + X_{ij} + X_{jt} \end{aligned}$$

The minimum travel time distribution is given by:

$$P^*_{st} = \mathcal{MIN}(P^1_{st}, P^2_{st}, P^3_{st}).$$

Using the cumulative distribution function with travel time realization c:

$$F_{P^*_{st}}(c) = 1 - \Pr(P^1_{st} > c, P^2_{st} > c, P^3_{st} > c)$$

and expanding:

$$F_{P^*_{st}}(c) = 1 - \Pr(X_{si} + X_{it} > c, X_{sj} + X_{jt} > c, X_{si} + X_{ij} + X_{jt} > c)$$

In general, this statement cannot be separated further due to the dependence among P_{st}^1, P_{st}^2 and P_{st}^3 . However, if X_{si} is fixed to a value of x_{si} and X_{jt} is fixed to a value x_{jt}, then all the remaining random variables become independent. In other words, we have:

$$F_{P_{st}^*}(c) = 1 - \Pr(X_{it} > c - x_{si}, X_{sj} > c - x_{jt}, X_{ij} > c - x_{si} - x_{jt} | x_{si}, x_{jt})$$

and, separating,

$$F_{P_{st}^*}(c) = 1 - \Pr(X_{it} > c - x_{si} | x_{si}, x_{jt}) \Pr(X_{sj} > c - x_{jt} | x_{si}, x_{jt}) \\ \Pr(X_{ij} > c - x_{si} - x_{jt} | x_{si}, x_{jt})$$

$$F_{P_{st}^*}(c) = 1 - \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{c-x_{si}}^{\infty} f_{X_{it}}(x_{it}) dx_{it} \int_{c-x_{jt}}^{\infty} f_{X_{sj}}(x_{sj}) dx_{sj} \\ \int_{c-x_{jt}-x_{si}}^{\infty} f_{X_{ij}}(x_{ij}) dx_{ij} \, f_{X_{jt}}(x_{jt}) f_{X_{si}}(x_{si}) dx_{si} dx_{jt} \quad (3.1)$$

This example shows that even in "simple" non-series-parallel graphs like the Wheatstone Bridge, the analytical calculations are quite complicated. These difficulties have led to a variety of approaches in the analysis of minimum travel time distributions on non-series-parallel graphs.

Building from the work on series-parallel graphs, Martin generalizes his approach to non-series-parallel graphs. A method is proposed to transform a non-series-parallel $\mathcal{TT}$-$\mathcal{DAG}$ to a series-parallel $\mathcal{TT}$-$\mathcal{DAG}$ by adding "dual" arcs. "Dual" arcs are duplicates of arcs that are not in series or in parallel. Analytically, this transformation is problematic since the "dual" arcs are not independent of each other. To solve this problem, Martin is forced to calculate the joint distributions of the "dual" arcs, and the joint distribution of all paths in each parallel subnet in the transformed series-parallel $\mathcal{TT}$-$\mathcal{DAG}$. In the worst case, the calculation resulting from the "dual" arc transformation becomes large and inefficient.

Hartley and Wortham develop an algorithm that reduces a network until a Wheatstone Bridge structure is found [18]. This Wheatstone Bridge can then be reduced according to Equation 3.1. After removing the Wheatstone Bridge, the algorithm continues until the network is either reduced entirely or another Wheatstone Bridge is found.

In [12], Frank considers the calculation of the minimum travel time distribution on directed graphs. Frank's approach is more general in that it does not assume arc travel time independence, acyclic graphs, or any kind of series-parallel reducibility. However, knowledge of the arc travel time joint distribution function is assumed. Frank proceeds by observing for a source s and sink t:

$$\vec{P} = \vec{X}^T \delta$$

where $\vec{P}$ is the one by $|\Pi_{st}|$ vector $\vec{P} = [P_{st}^1, P_{st}^2, P_{st}^3, \ldots]$ of the travel time distributions

for each path. δ is a m by $|\Pi_{st}|$ indicator matrix with each entry $\delta_{lk} \in \{0, 1\}$ depending on whether the lth arc (i, j) is part of the kth path P_{st}^k. $\vec{X}$ is a m by one vector of the arc travel time distributions (and $\vec{X}^T$ is its transpose). Note that $P_{st}^* = \mathcal{MIN}(\vec{P})$. Frank notes this large computation can be reduced by using characteristic functions. The resulting calculation (as a characteristic function) is then a multi-variable integral in m-dimensions. The problem with this approach is that the calculation can become very time-consuming and the inversion of the final characteristic function must still be performed. The difficulty of the inversion depends on the rank of δ. After this calculation is made, the final calculation of $P_{st}^* = \mathcal{MIN}(P_{st}^1, P_{st}^2, P_{st}^3, \ldots)$ is performed. Frank and Hakimi have a preceding paper that also outlines this approach with random arc capacities rather than random arc travel times [13].

Because of the above difficulties, Frank resorts to Monte Carlo simulation to obtain the minimum travel time distribution. *Straight-forward* or *Crude* Monte-Carlo sampling can be used to approximate the cumulative distribution function, $F_{P_{st}^*}$ of P_{st}^*. If each arc travel time random variable X_{ij} is fixed to a particular realization x_{ij}, let π_{st}^* be the deterministic fastest path associated with the particular realizations x_{ij}. For a given value l, Frank defines the function $\lambda(l, \pi_{st}^*)$ as:

$$\lambda(l, \pi_{st}^*) = \begin{cases} 1 & \text{if } \sum_{(i,j)\in\pi_{st}^*} x_{ij} \leq l \\ 0 & otherwise \end{cases}$$

If each arc (i, j) is sampled and fixed to a x_{ij} C times, C fastest paths can be calculated. For a particular value l, $F_{P_{st}^*}(l)$ is then approximated by:

$$F_{P_{st}^*}(l) = \frac{1}{C} \sum_{c=1}^{C} \lambda(l, \pi_{st}^*(c))$$

where $\pi_{st}^*(c)$ is the travel time of the fastest path for the cth set of arc realizations. The cumulative distribution $F_{P_{st}^*}$ is then differentiated to obtain $f_{P_{st}^*}$.

Alternatively, $f_{P_{st}^*}$ can be calculated directly by sampling values for each arc, running a deterministic fastest path algorithm, and recording the value of the fastest path. If this process is repeated, an approximation of $f_{P_{st}^*}$ can be obtained.

Frank also covers two statistical based methods, but most related work builds off Monte Carlo sampling or Martin's series-parallel work.

Burt and Garman make the observation that Monte Carlo sampling every arc is not necessary [20]. They present a method referred to as *Conditional Monte Carlo* where one only need sample the "non-unique" arcs. For a given source s and sink t, non-unique arcs are arcs that are contained in more than one path from s to t. When the non-unique arcs are conditioned on Monte Carlo samples, the (s, t)-paths in the network become independent. Consequently, in this conditioned network, the conditional minimum travel time distribution can be calculated by taking the sum and the minimum of independent travel time random variables. The unconditional minimum travel time distribution can then be estimated by iterating over a set of Monte Carlo samples. Burt and Garman report that Conditional Monte Carlo sampling produced

a better estimate of the correct distribution on the Wheatstone Bridge (Figure 3-1) than Crude Monte Carlo sampling.

To illustrate Conditional Monte Carlo sampling, consider Figure 3-1. The non-unique arcs are (s,i) and (j,t). Conditional Monte Carlo proceeds by conditioning on each of these arcs and computing $P^*_{st} = \mathcal{MIN}(P^1_{st}, P^2_{st}, P^3_{st})$, where:

$$\begin{aligned} P^1_{st} &= x_{si} + X_{it} \\ P^2_{st} &= X_{sj} + x_{jt} \\ P^3_{st} &= x_{si} + X_{ij} + x_{jt} \end{aligned}$$

Conditional Monte Carlo can be thought of as a middle ground between Crude Monte Carlo sampling and a direct analytical solution.

Garman adds a follow-on work in [17] to even further reduce the number of Monte Carlo samples required. Garman observes that by conditioning on certain arcs, a network can be reduced to a series-parallel graph where the (conditional) minimum travel time distribution can be calculated using Martin's algorithm. Furthermore, Garman provides a theorem that states a property of series-parallel reducibility.

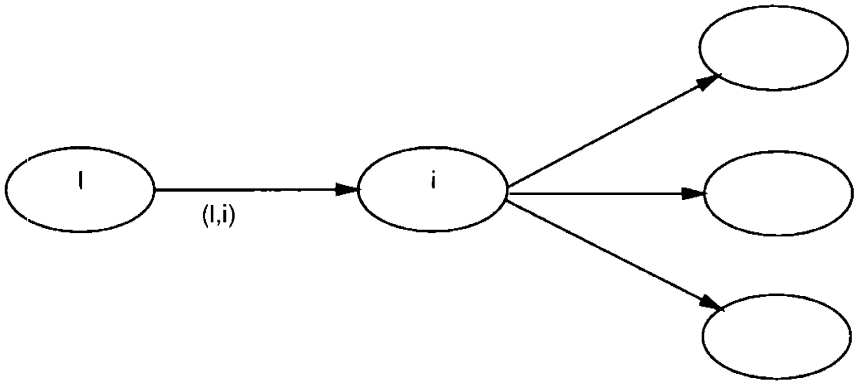
Theorem 3 (Properties of Series-Parallel Reducible Graphs). *A $\mathcal{TT}$-$\mathcal{DAG}$ G' which cannot be reduced to a single arc via series and parallel reductions will possess (1) at least one arc (l,i) that has more than one successor while each of (l,i)'s successors has only (l,i) as its predecessor, and (2) at least one arc (i,j) that has more that one predecessor and each of (i,j)'s predecessors has only (i,j) as its successor.*

The arcs in part (1) of Theorem 3 are referred to as Type-1 arcs and the arcs in part (2) of Theorem 3 are referred to as Type-2 arcs. Figure 3-2(a) shows a Type-1 arc and Figure 3-2(b) shows a Type-2 arc. Using this terminology, the proof of Theorem 3, as presented in [17], is:

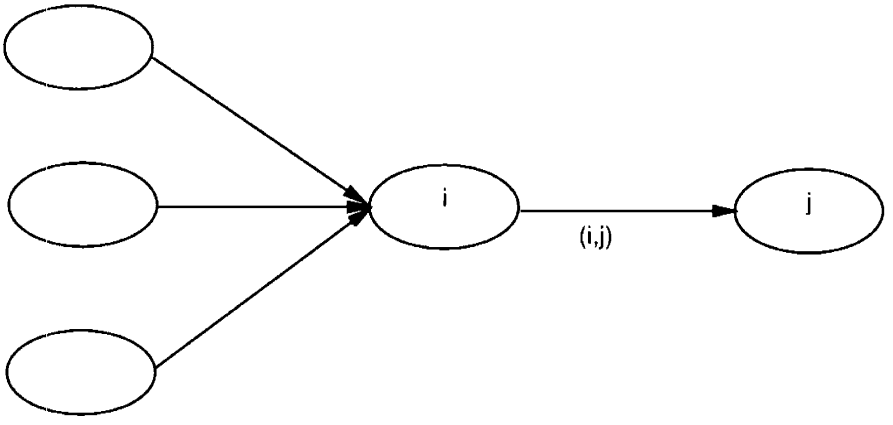
Proof. Given a $\mathcal{TT}$-$\mathcal{DAG}$ G' which is not series-parallel reducible and a source s and sink t. Consider all outgoing arcs $\{(s,j)|j \in N^+(s)\}$ from the source s. It must be the case that a Type-1 arc exists. Otherwise, a parallel reduction is possible or the network is cyclic, both of which are not true. Next, consider all incoming arcs to the sink t. It must be the case that a Type-2 arc exists. Otherwise, a parallel reduction is possible or the network is cyclic, both of which are not true. □

Using Theorem 3, Garman presents an algorithm that calculates a conditional minimum travel time distribution $f_{P^*_{st}|(x_{ij}\ldots)}$ where $(x_{ij}\ldots)$ represents the arcs that are conditioned upon. The algorithm is listed in Algorithm 3.1. After $f_{P^*_{st}|(x_{ij}\ldots)}$ is calculated, $f_{P^*_{st}}$ can be calculated by Monte Carlo simulation over the conditioned arcs.

Approximately twenty years after Garman's work, Bein et al. address the same subject with a more theoretical approach [1]. Bein et al. independently come to same conclusions. One concept that Bein et al. define is $s \rightarrow t$ $\mathcal{DAG}$ *complexity*, which is a measure of how "series-parallel" a $\mathcal{TT}$-$\mathcal{DAG}$ is. Technically, the $s \rightarrow t$ $\mathcal{DAG}$



(a) (l, i) is a Type-1 Arc.



(b) (i, j) is a Type-2 Arc.

Figure 3-2: Type-1 and Type-2 Arcs.

Input: G' is a $\mathcal{TT}$-$\mathcal{DAG}$. $G' = (N', A')$. $s, t \in N'$. $X = \{X_{ij} | (i,j) \in A'\}$.
Output: $f_{P^*_{st}|(x_{ij}...)}$.
  Step 0: Initialization.
  while $|A'| \neq 1$ **do**
    Step 1: Perform any possible series and parallel reductions.
    Step 2: Find a Type-1 or Type-2 arc and condition on it.
    if $|A'| \neq 1$ **then**
      if a Type-1 arc (l,i) is found **then**
        for $(i,j) \in A^+(i)$ **do**
          $A' = A' \cup (l,j)$
          $X_{lj} = x_{li} + X_{ij}$
          $A' = A' \setminus (i,j)$
        $A' = A' \setminus (l,i)$
      else if a Type-2 arc (i,j) is found **then**
        for $(l,i) \in A^-(i)$ **do**
          $A' = A' \cup (l,j)$
          $X_{lj} = X_{li} + x_{ij}$
          $A' = A' \setminus (l,i)$
        $A' = A' \setminus (i,j)$

Algorithm 3.1: Garman's Conditional Sampling Algorithm.

complexity of a graph G is the minimum number of nodes (and adjacent arcs) that need to be removed to make G' series-parallel reducible. They present an algorithm to find the $s \rightarrow t$ $\mathcal{DAG}$ complexity of G'. This algorithm is very similar to Garman's algorithm, except the graph in question is first transformed to an "auxiliary graph". The auxiliary graph is a graph that identifies all embedded Wheatstone Bridges. From the auxiliary graph, a minimum vertex cover is calculated and is input into what is equivalent to Garman's algorithm.

Kulkarni considers the special case where arc costs are independent and exponentially distributed [23]. A continuous time Markov chain is constructed. The key observation in this work is that the set of minimum cuts in the network corresponds to the state-space of the Markov chain. Using this approach, Kulkarni presents algorithms to calculate the shortest path distribution and the associated moments. This approach suffers from exponential state-space growth as the size of the network increases.

Robillard and Trahan [36] present a bounding approach for the distribution of the project duration time in PERT/CPM networks. Rather than concern themselves with the dependence between paths, Robillard and Trahan observe that by ignoring dependence considerations, lower-bounds can be derived for the project completion time distribution using Bonferroni inequalities. If H_i are events such that $H = \cup_i H_i$,

then the Bonferroni inequalities are:

$$\begin{aligned}
\Pr(\cup H_i) &\leq \sum_i \Pr(H_i) \\
\Pr(\cup H_i) &\geq \sum_i \Pr(H_i) - \sum_{i<j} \Pr(H_i \cap H_j) \\
\Pr(\cup H_i) &\leq \sum_i \Pr(H_i) - \sum_{i<j} \Pr(H_i \cap H_j) + \sum_{i<j<k} \Pr(H_i \cap H_j \cap H_k)
\end{aligned}$$

One can use the Bonferroni inequalities to derive an upper-bound for the minimum travel time distribution. Robillard and Trahan compare their bounds to previous analytical calculations and find decent results. Similar ideas and approaches are also presented in [21].

In [14], an in-depth analysis is provided for networks where the arc travel times are independent and identically distributed. Both Sigal et al. [39] and Fishman [11] calculate the minimum travel time distribution via a cut-set approach. Elmaghraby [9] provides an excellent summary of the issues and approaches in PERT/CPM networks. Aside from covering the material in this section, Elmaghraby also discusses antithetic variates and control variates, which are two statistical methods. These statistical methods utilize positive and negative correlation to calculate the project completion time distribution.

3.5 Two-Terminal Directed Acyclic Graph Generation

As noted at the end of Chapter 2, it may not be necessary to consider all of G in the calculation of P^*_{st}. Instead, we should only consider a $\mathcal{TT}$-$\mathcal{DAG}$ subgraph G' of G whose arcs have some potential to be used in the fastest path from node s to node t. This problem is exaggerated in probabilistic networks, where it is more difficult to determine if a particular arc (i, j) will be used in a fastest path.

One way to generate such a $\mathcal{TT}$-$\mathcal{DAG}$ G' in a deterministic network is to take some union of the k shortest paths from node s to node t. The k shortest paths problem is a well studied variant of the single shortest path problem (see [10]). Within the general k shortest paths problem, there are two major variants: the k shortest *simple* paths and the k (possibly cyclic) shortest paths. The former problem is more difficult to solve than the latter problem. In [10], Eppstein studies the k shortest paths problem when the paths are not restricted to be simple. An efficient algorithm is presented and its impact on several applications is studied. The primary result in the k shortest simple paths problem is an efficient algorithm in k due to Yen [47]. In the context of this thesis, note that G' is not necessarily acyclic if G' is generated by the simple union of the k shortest simple paths from node s to node t.

In [7], Dial considers methods to calculate multiple (s, t)-paths in a deterministic transportation network. Dial's motivation is to avoid the congestion problems

associated with an all-or-nothing traffic assignment. To do this without performing an inefficient path enumeration, Dial discusses an approach for the identification of "efficient paths". Dial's definition of an efficient path is a path where every arc (i, j) has node i closer to source s than node j and node j closer to sink t than node i. A consequence of this approach is that, for a given source s and sink t, Dial's algorithm generates a $\mathcal{TT}$-$\mathcal{DAG}$ subgraph G' of G where each arc in G' is part of at least one efficient path from s to t. The five stated goals of Dial's algorithm are:

1. All efficient paths from node s to node t should have a positive probability of being used.

2. All efficient paths of equal travel time should have an equal probability of being used.

3. Between two efficient paths with unequal travel times, the faster path should have a higher probability of being used.

4. The user of the model should have some control over the probabilities.

5. Path enumeration should be avoided.

Dial's algorithm achieves these goals by working with arcs rather than paths. The general idea is to assign each arc (i, j) a likelihood, $L(i, j)$, of being used in a fastest path from source s to sink t. The likelihood of an arc (i, j) is calculated using the (deterministic) travel times from source s to nodes i and j, and from nodes i and j to sink t. Dial conjectures that the probability of a using a path is proportional to the product of the likelihoods of the arcs in the path. However, Dial does not explicitly construct the efficient paths. Instead, a graph is built where every path is efficient.

The algorithm initializes by running two algorithms; a one-to-all fastest paths algorithm from source s and an all-to-one fastest paths algorithm to sink t. The fastest paths labels from source s to node i are denoted d_{si} and the fastest paths labels to sink t from node i are denoted d_{it}. Using these travel time labels, each arc is assigned a likelihood $L(i, j)$ using the following equation:

$$L(i,j) = \begin{cases} \exp^{\theta(d_{sj} - d_{si} + x_{ij})} & \text{if } d_{si} < d_{sj}\,, d_{jt} < d_{it} \\ 0 & otherwise \end{cases}$$

where x_{ij} is the deterministic travel time of arc (i, j). Note that by simply taking all arcs with $L(i, j) > 0$ we are not guaranteed a $\mathcal{TT}$-$\mathcal{DAG}$. In the next step of the algorithm, Dial constructs the graph of efficient paths by calculating arc "weights". The weights are used to recursively construct G', the graph of efficient paths. The arc weight $w(i, j)$ is calculated by traversing each arc (i, j) in increasing order of d_{si}:

$$w(i,j) = \begin{cases} L(i,j) & \text{if } i = s \\ L(i,j) \sum_{(u,i)|u \in N^-(i)} w(u,i) & otherwise \end{cases}$$

At this point, if we take all arcs (i,j) with $w(i,j) > 0$ we will have a $\mathcal{TT}$-$\mathcal{DAG}$ of efficient paths. Since Dial is concerned with assigning traffic to the arcs, an additional step is performed. The traffic volumes $v(i,j)$ are assigned in descending order of d_{sj} for each arc (i,j):

$$v(i,j) = \begin{cases} \frac{y \cdot w(i,j)}{\sum_{(u,j)|u \in N^-(j)} w(u,j)} & \text{if } j = t \\ \frac{w(i,j) \sum_{(j,l)|l \in N^+(j)} v(j,l)}{\sum_{(u,j) \in N^-(j)} w(u,j)} & otherwise \end{cases}$$

where y is the demand from source s to sink t.

For the purpose of this thesis, the key contribution of Dial's algorithm is that it constructs a $\mathcal{TT}$-$\mathcal{DAG}$ G' from a (possibly cyclic) graph G without path enumeration. Furthermore, using Dial's definition, G' is composed entirely of efficient paths.

Dial also discusses an alternative, but similar, definition for an efficient path. In the alternative case, an efficient path is defined as a path composed of arcs (i,j) where node i is closer to source s than node j. This definition effectively changes the algorithm from a (s,t)-path algorithm to a "s-to-all" algorithm. The calculation of the likelihoods and the arc weights remains the same. However, because it is now a one-to-all algorithm, we cannot construct a $\mathcal{TT}$-$\mathcal{DAG}$ from source s to sink t by simply taking $\{(i,j)|w(i,j) > 0\}$. The final step of traffic assignment is different since there are multiple sinks. We omit this final step as it is not relevant to the work in this thesis.

As an extension to the algorithm, Dial also suggests a method to calculate the probability distribution that a path from source s to sink t takes a discrete travel time l. Although this is similar to the notion of the minimum travel time distribution, it is a substantially less complex problem since the arc travel times are deterministic.

Let $p_{\pi_{st}}(l)$ be the probability that a path from source s to sink t takes a discrete value of l. Dial determines $p_{\pi_{st}}(l)$ by recursively calculating $p_{\pi_{sj}}(l)$. Each node j is selected in ascending order of d_{sj} until $j = t$. The steps are as follows:

1. (Initialization) $p_{\pi_{ss}}(0) = 1$. $p_{\pi_{ss}}(l) = 0 \; \forall$ discrete path travel times $l > 0$
2. (Recursive) $p_{\pi_{sj}}(l) = \sum_{(i,j)|i \in N^-(j)} p_{\pi_{si}}(l - x_{ij}) \frac{w(i,j)}{\sum_{(u,j)|u \in N^-(j)} w(u,j)}$

[This page intentionally left blank.]

Chapter 4

Minimum Travel Time Distribution Algorithms

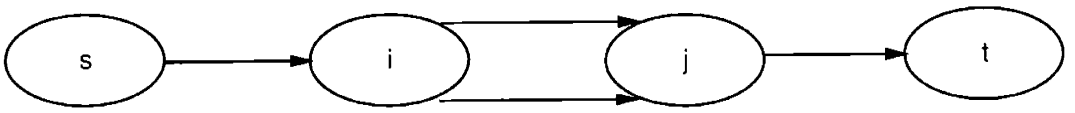
In this chapter, we describe how to calculate the minimum travel time distribution in a probabilistic network. As noted, the concept of a series-parallel network is fundamental to this calculation. First, we discuss how to determine if a graph is series-parallel reducible. If a graph is series-parallel reducible, the minimum travel time distribution can be calculated exactly and efficiently. If the graph is not series-parallel, the minimum travel time distribution can be approximated in an efficient manner.

4.1 Introduction

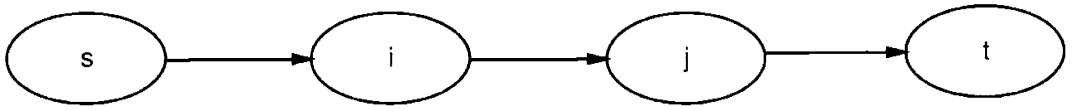
In this chapter, we assume we are given a $\mathcal{TT}$-$\mathcal{DAG}$, G', with source s and sink t. As noted in Chapter 2, we also assume the arc travel time random variables, X_{ij}, are independent. If G' is series-parallel, then we proceed with an algorithm that is based upon the work of Lawler, Tarjan, and Valdes (see Chapter 3). If G' is not series-parallel, then we proceed with an algorithm primarily based on Algorithm 3.1 (see Chapter 3). This basic procedure is outlined in Algorithm 4.1.

Input: $G' = (N', A')$. G' is a $\mathcal{TT}$-$\mathcal{DAG}$. $s, t \in N'$.
Output: P^*_{st}.
  Step 1: Determine if G' is series-parallel.
  if G' is series-parallel **then**
    Step 2a: $P^*_{st} \leftarrow$ Use an exact series-parallel method.
  else
    Step 2b: $P^*_{st} \leftarrow$ Use an approximation method.

Algorithm 4.1: Basic Procedure for Calculating P^*_{st}.



(a) A Series-Parallel Network.



(b) Figure 4-1(a) After a Parallel Reduction.

Figure 4-1: Order of operations affects independence. By choosing an appropriate order of series and parallel reductions in Figure 4-1(a), we maintain independence among travel time random variables. For example, by performing the parallel reduction $P_{ij} = \mathcal{MIN}(X_{ij}^1, X_{ij}^2)$ first, we get Figure 4-1(b) where the arc travel time random variables are still independent. See Section 4.2.

4.2 Series-Parallel Graphs

To build intuition for series-parallel graphs, we consider the calculation of P_{st}^* in Figure 4-1(a). There are two different ways to calculate P_{st}^*. In the first approach, we calculate:

$$P_{st}^* = \mathcal{MIN}(X_{si} + X_{ij}^1 + X_{jt}, X_{si} + X_{ij}^2 + X_{jt}) \tag{4.1}$$

Note that even if $X_{si}, X_{ij}^1, X_{ij}^2, X_{jt}$ are independent, the path travel time random variables $P_{st}^1 = X_{si} + X_{ij}^1 + X_{jt}, P_{st}^2 = X_{si} + X_{ij}^2 + X_{jt}$ are not. Since the arguments to the $\mathcal{MIN}$ function are not independent, we cannot easily evaluate it. However, we can also evaluate P_{st}^* as:

$$P_{st}^* = X_{si} + \mathcal{MIN}(X_{ij}^1, X_{ij}^2) + X_{jt} \tag{4.2}$$

We evaluate Equation 4.2 by performing the parallel reduction $P_{ij} = \mathcal{MIN}(X_{ij}^1, X_{ij}^2)$ first. Note that the arguments to the $\mathcal{MIN}$ function are independent and, consequently, we can evaluate it as described in Section 2.4.2. The parallel reduction removes arcs $(i,j)^1, (i,j)^2$ and adds a new arc (i,j) with $X_{ij} = \mathcal{MIN}(X_{ij}^1, X_{ij}^2)$ as the arc travel time. Figure 4-1(b) is the graph after the parallel reduction. In this graph, all arcs are independent and in series. Therefore, we can calculate $P_{st}^* = X_{si} + X_{ij} + X_{jt}$.

Both Equation 4.1 and Equation 4.2 calculate the same random variable, P_{st}^*. However, by choosing the correct *order of operations* the independence among the random variables is *maintained*, and Equation 4.2 becomes easier to evaluate than Equation 4.1. In fact, the order of operations forms a non-unique binary tree [43].

The graph in Figure 4-1(a) is easy to evaluate because it is series-parallel re-

ducible. In the case of a non-series parallel graph like the Wheatstone Bridge, we cannot maintain independence. However, if we can efficiently identify series-parallel subgraphs of non-series-parallel graphs, we can reduce the complexity of calculating P_{st}^* in non-series-parallel graphs.

4.2.1 Testing for Series-Parallel Reducibility

To test whether or not a graph is series-parallel reducible, we use a slightly modified version of the algorithm presented in [44], [43]. The modifications are only concerned with the choice of data structures and do not affect the general approach of [44], [43].

The algorithm is presented in Algorithm 4.2. The key data structure in this algorithm is the queue, denoted Q. The algorithm initializes by adding all $i \in N', i \neq s, i \neq t$ to Q. At each iteration, a node i is selected from the front of Q via the queue operation $pop\_front()$. Next, we perform incoming and outgoing parallel reductions with node i as the tail and head node respectively.

Note that we "short-circuit" the removal of outgoing (incoming) parallel arcs if node i has at least two distinct successors (predecessors). In other words, we stop processing the outgoing (incoming) parallel arcs of node i if $|N^+(i)| \geq 2$ ($|N^-(i)| \geq 2$) since we know we will not be able to remove node i with a series reduction at this time. Therefore, node i may still be incident with parallel arcs at the end of each iteration. As long as G' is series-parallel reducible, the parallel arcs will be removed at a later time. Note that the short-circuit is not part of the algorithm from [44], [43]. However, it allows for run-time improvement since we only remove parallel arcs when necessary. Also note that the short-circuit does not affect the correctness of the algorithm; it only changes the order of the parallel reductions.

In the context of the algorithm, the parallel reduction is implemented by removing all parallel arcs *except the first arc* for a given pair of nodes i and j. If, after these parallel reductions, $|N^-(i)| = 1$ and $|N^+(i)| = 1$, then a series reduction is possible. In this case, node i is removed from G', along with the incoming arc (l, i) and the outgoing arc (i, j); arc (l, j) is added. Nodes l and j are added to Q via the $enqueue()$ operation, provided $l \neq s, j \neq s, l \neq t, j \neq t$ and nodes l, j are not already in Q. If a series reduction is not possible, the next iteration begins and a new node selected. As long as G' is series-parallel reducible, then node i will be removed from the graph in a later iteration.

Once Q becomes empty, we can check to see if the graph is series-parallel by checking $|N'|$. If $|N'| = 2$ then the two remaining nodes must be s and t since s and t are never added to Q. Any remaining arcs can only take the form of (s, t) since G' is assumed to be a $\mathcal{TT}$-$\mathcal{DAG}$. If $|A(s, t)| > 1$, we could perform more parallel reductions, but this is inconsequential; G' has effectively been transformed to a single arc by series and parallel reductions and thus, must be series-parallel reducible.

To see the correctness of this algorithm we consider the two possible results of Algorithm 4.2. In the case that $TRUE$ is returned, then we know there are two remaining nodes in G' and, since nodes s and t are never added to Q, these nodes must be s and t. In the case where $FALSE$ is returned, we know there are more than two nodes in G', each of which has either at least two distinct predecessors

Input: $G' = (N', A')$. G' is a $\mathcal{TT}$-$\mathcal{DAG}$. $s, t \in N'$.
Output: Return $TRUE$ if G' is series-parallel reducible.

```
Step 0: Initialize by adding all nodes i ∈ N', i ≠ s, t to Q.
for i ∈ N' do
  if i ≠ s AND i ≠ t then
    Q.enqueue(i)
while Q.length() ≥ 1 do
  i ← Q.pop_front()
  Step 1: If possible, remove all parallel arcs with i as the head node.
  if |N+(i)| < 2 then
    for j ∈ N+(i) do
      for k ∈ 2 ... |A(i, j)| do
        A' ← A' \ (i, j)^k
  Step 2: If possible, remove all parallel arcs with i as the tail node.
  if |N-(i)| < 2 then
    for j ∈ N-(i) do
      for k ∈ 2 ... |A(j, i)| do
        A' ← A' \ (j, i)^k
  Step 3: If possible, perform a series reduction.
  if N-(i) = 1 AND N+(i) = 1 then
    j ← N+(i)
    l ← N-(i)
    A' ← A' \ (i, j)
    A' ← A' \ (l, i)
    A' ← A' ∪ (l, j)
    N' ← N' \ i
    if l ∉ Q AND l ≠ s AND l ≠ t then
      Q.enqueue(l)
    if j ∉ Q AND j ≠ s AND j ≠ t then
      Q.enqueue(j)
if |N'| = 2 then
  return TRUE
else
  return FALSE
```

Algorithm 4.2: Determining if a Graph is Series-Parallel Reducible.

$(N^-(i) \geq 2)$ or at least two distinct successors $(N^+(i) \geq 2)$. The latter point implies that the graph cannot be reduced further, and is therefore not series-parallel reducible.

The finiteness of the algorithm is seen by analyzing the size of Q and, thus, counting the number of possible iterations of the loop. Let $n = |N'|$ and $m = |A'|$. Initially, $n - 2$ nodes are added to Q. Parallel reductions do not add any nodes to Q, but series reductions potentially add two nodes to Q. Thus, the maximum size of Q is bounded by $(n-2) + 2(n-2)$ and, consequently, the algorithm is finite. The number of nodes removed from Q will be at most $(n-2)+2(n-2)$. There are m arcs in A' initially. Although arcs are not added with the parallel reduction, the series reduction does add a single arc. However, the series reduction also removes two arcs, for a total decrease of m by one. Therefore, the total time we spend removing arcs from A' is bounded by $O(m)$. Combining this with the total time we spend removing nodes from N' and we see the complexity of this algorithm is $O(n+m)$.

4.3 Minimum Travel Time Distributions on Series-Parallel Graphs

If G' is series-parallel reducible, then the minimum travel time distribution can be calculated according to Algorithm 4.3. This algorithm is essentially the same as Algorithm 4.2 with the only effective modification being the addition of the sum and the minimum functions.

Although we still check for $|N^-(i)| < 2$ and $|N^+(i)| < 2$, it is of no practical consequence since G' is series-parallel (see Section 4.2.1). It is only included here to illustrate the similarity to Algorithm 4.2.

Since the only additions to Algorithm 4.2 are the sum and the minimum of random variables, the calculation of P^*_{st} can be achieved efficiently as long as these functions can be implemented efficiently. Let the complexity of the sum of random variables be denoted $\overline{C}$ and let the complexity of the minimum of random variables be denoted $\overline{M}$. Then the complexity of Algorithm 4.3 is $O(n\overline{C} + m\overline{M})$. We assume the sum and the minimum routines run in pseudo-polynomial time, and we analyze the complexity of our implementation of these routines in Chapter 5.

In light of Algorithm 4.3, we have the following result, which shows that for a given graph G', P^*_{st} is as good as any path from source s to sink t in terms of expectation.

Theorem 4. *Let P_{st} represent the travel time of a path π_{st} from source s to sink t in a series-parallel graph G'. Then $E[P^*_{st}] \leq E[P_{st}]\ \forall \pi_{st} \in \Pi_{st}$ where P^*_{st} is calculated according to Algorithm 4.3.*

Proof. Algorithm 4.3 is composed of two functions: the sum and the minimum of random variables. π_{st} is necessarily composed of arcs that will be identified as arcs in series or arcs in parallel by Algorithm 4.3. Therefore, to show $E[P^*_{st}] \leq E[P_{st}]$, it is enough to show 1) $E[X_{ik}] \leq E[X_{ij}] + E[X_{jk}]$ for two arcs $(i,j),(j,k)$ in series and 2) $E[X_{ij}] \leq E[\mathcal{MIN}(X^1_{ij}, X^2_{ij})]$ for arcs $(i,j)^1, (i,j)^2$ in parallel.

First, when $X_{ik} = X_{ij} + X_{jk}$, $E[X_{ik}] = E[X_{ij}] + E[X_{jk}]$ by linearity of expectation.

Input: $G' = (N', A')$. G' is a series-parallel $\mathcal{TT}$-$\mathcal{DAG}$. $s, t \in N'$.
Output: P^*_{st} .

Step 0: Initialize by adding all nodes $i \in N', i \neq s, t$ to Q.
for all $i \in N'$ **do**
  if $i \neq s$ AND $i \neq t$ **then**
    $Q.enqueue(i)$
while $Q.length() \geq 1$ **do**
  $i \leftarrow Q.pop\_front()$
  Step 1: If possible, remove all parallel arcs with i as the head node.
  if $|N^+(i)| < 2$ **then**
    for all $j \in N^+(i)$ **do**
      $X^*_{ij} \leftarrow X^1_{ij}$
      for all $k \in 2 \ldots |A(i,j)|$ **do**
        $X^*_{ij} \leftarrow \mathcal{MIN}(X^k_{ij}, X^*_{ij})$
        $A' \leftarrow A' \setminus (i,j)^k$
      $X_{ij} \leftarrow X^*_{ij}$
  Step 2: If possible, remove all parallel arcs with i as the tail node.
  if $|N^-(i)| < 2$ **then**
    for all $j \in N^-(i)$ **do**
      $X^*_{ji} \leftarrow X^1_{ji}$
      for all $k \in 2 \ldots |A(j,i)|$ **do**
        $X^*_{ji} \leftarrow \mathcal{MIN}(X^k_{ji}, X^*_{ji})$
        $A' \leftarrow A' \setminus (j,i)^k$
      $X_{ji} \leftarrow X^*_{ji}$
  Step 3: If possible, perform a series reduction.
  if $N^-(i) = 1$ AND $N^+(i) = 1$ **then**
    $j \leftarrow N^+(i)$
    $l \leftarrow N^-(i)$
    $A' \leftarrow A' \setminus (i,j)$
    $A' \leftarrow A' \setminus (l,i)$
    $N' \leftarrow N' \setminus i$
    $X_{lj} \leftarrow X_{li} + X_{ij}$
    $A' \leftarrow A' \cup (l,j)$
    if $l \notin Q$ AND $l \neq s$ AND $l \neq t$ **then**
      $Q.enqueue(l)$
    if $j \notin Q$ AND $j \neq s$ AND $j \neq t$ **then**
      $Q.enqueue(j)$
Step 4: Remove all remaining parallel arcs from s to t.
$X^*_{st} \leftarrow X^1_{st}$
for all $k \in 2 \ldots |A(s,t)|$ **do**
  $X^*_{st} \leftarrow \mathcal{MIN}(X^k_{st}, X^*_{st})$
  $A' \leftarrow A' \setminus (s,t)^k$
$P^*_{st} \leftarrow X^*_{st}$

Algorithm 4.3: Minimum Travel Time Distribution on a Series-Parallel Graph.

When $X_{ij} = \mathcal{MIN}(X_{ij}^1, X_{ij}^2)$, we can write the cumulative distribution function as $F_{X_{ij}}(c) = F_{X_{ij}^1}(c) + F_{X_{ij}^2}(c) - F_{X_{ij}^1}(c)F_{X_{ij}^2}(c)$. If we rewrite this as $F_{X_{ij}}(c) = F_{X_{ij}^1}(c) + F_{X_{ij}^2}(c)(1 - F_{X_{ij}^1}(c))$, we see that $F_{X_{ij}}(c) \geq F_{X_{ij}^1}(c)$ if $F_{X_{ij}^1}(c) \geq F_{X_{ij}^2}(c)$, and $F_{X_{ij}}(X_{ij}) \geq F_{X_{ij}^2}(c)$ if $F_{X_{ij}^1}(c) \leq F_{X_{ij}^2}(c)$. Another way to see this is to observe that since $F_{X_{ij}^2}(c)(1 - F_{X_{ij}^1}(c)) \geq 0$ then $F_{X_{ij}}(c) \geq F_{X_{ij}^1}(c)$. If we rewrite $F_{X_{ij}}(c) = F_{X_{ij}^1}(c) + F_{X_{ij}^2}(c) - F_{X_{ij}^1}(c)F_{X_{ij}^2}(b)$ as $F_{X_{ij}}(c) = F_{X_{ij}^2}(c) + F_{X_{ij}^1}(c)(1 - F_{X_{ij}^2}(c))$ then by similar reasoning we have $F_{X_{ij}}(c) \geq F_{X_{ij}^2}(c)$. The consequence of these observations is that $E[X_{ij}] \leq E[X_{ij}^1]$ and $E[X_{ij}] \leq E[X_{ij}^2]$. Therefore, we conclude that $E[P_{st}^*] \leq E[P_{st}] \, \forall \pi_{st} \in \Pi_{st}$. □

4.4 Minimum Travel Time Distributions on Non-Series-Parallel Graphs

Garman's theorem (Theorem 3) provides the framework for an efficient approximation of P_{st}^* in non-series-parallel graphs. The algorithm presented in Algorithm 4.4 is composed of Algorithm 4.3 and Algorithm 3.1 from Chapter 3. At each iteration, the algorithm first calls a modified version of Algorithm 4.3 to perform series and parallel reductions on G'. Since G' is non-series-parallel, the modification is the removal of **Step 4**, which is the final calculation of P_{st}^*. After **Step 1** of Algorithm 4.4, there are two possible cases. In **Step 2a**, G' consists of only the two nodes, s and t, and parallel arcs from s to t. P_{st}^* is calculated and returned. This implies that in the alternative **Step 2b**, G' cannot be reduced to a single arc with series and parallel reductions. Following Algorithm 3.1, we condition on an arc. This is done by finding and conditioning on a Type-1 or Type-2 arc (i, j). In Garman's original work, arc (i, j) is conditioned on a Monte Carlo sample. In our work, we instead condition on the value $E[X_{ij}]$. This is done for three reasons:

- In the larger context of routing, we are concerned with the overall computational overhead. Monte Carlo sampling can be time intensive and thus, inefficient in practical settings.
- Conditioning with $E[X_{ij}]$ is straight-forward.
- By conditioning with $E[X_{ij}]$, we can guarantee $E[P_{st}^*] \leq E[P_{st}] \, \forall \pi_{st} \in \Pi_{st}$. See Theorem 5. This is the same result as with series-parallel graphs.

The finiteness and efficiency of **Step 1** and **Step 2a** has already been covered. Let $m = |A'|$ and $n = |N'|$. In **Step 2b**, m is decreased by one. Therefore, Algorithm 4.4 will terminate within a finite number of iterations. In the worse case, **Step 2b** must iterate through all n nodes to find a suitable node i. For each node i, it takes $O(m)$ to find a Type-1 arc (or Type-2 arc). Once a suitable node i is found, $O(m)$ arcs are added and removed from G'. Therefore, we can bound **Step 2b** in $O(nm)$. As **Step 1** and **Step 2** are efficient, this means Algorithm 4.4 is also efficient.

Input: $G' = (N', A')$. G' is a non-series-parallel $\mathcal{TT}$-$\mathcal{DAG}$. $s, t \in N'$.
Output: P^*_{st}.

loop

Step 1: Perform as many series and parallel reductions as possible.

$G' \leftarrow$ Call Algorithm 4.3 on G' (without **Step 4**)

if $|N'| = 2$ **then**

Step 2a: If the graph is reduced, return P^*_{st}.

$X^*_{st} \leftarrow X^1_{ij}$

for all $k \in 2 \ldots |A(s,t)|$ **do**

$X_{st} \leftarrow \mathcal{MIN}(X_{st}, X^*_{st})$

$A' \leftarrow A' \setminus (s,t)^k$

$P^*_{st} \leftarrow X^*_{st}$

return P^*_{st}

else

Step 2b: Find a Type-1 arc or a Type-2 arc and condition on it.

for all $i \in N'$ **do**

if $|N^+(i)| > 1$ AND $|N^-(i)| = 1$ AND $\forall j \in N^+(i), |N^-(j)| = 1$ **then**

$l \leftarrow N^-(i)$

for all $(i,j) \in A^+(i)$ **do**

$A' \leftarrow A' \cup (l,j)$

$X_{lj} \leftarrow E[X_{li}] + X_{ij}$

$A' \leftarrow A' \setminus (i,j)$

$A' \leftarrow A' \setminus (l,i)$

break

else if $|N^+(i)| = 1$ AND $|N^-(i)| > 1$ AND $\forall l \in N^-(i), |N^+(l)| = 1$ **then**

$j \leftarrow N^+(i)$

for all $(l,i) \in A^-(i)$ **do**

$A' \leftarrow A' \cup (l,j)$

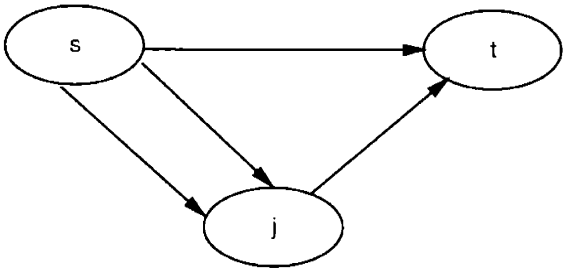
$X_{lj} \leftarrow X_{li} + E[X_{ij}]$

$A' \leftarrow A' \setminus (l,i)$

$A' \leftarrow A' \setminus (i,j)$

break

Algorithm 4.4: Minimum Travel Time Distribution on a Non-Series-Parallel Graph.



(a) The Resulting Graph.

Arc	Travel Time
$(s,j)^1$	$E[X_{si}] + X_{ij}$
$(s,j)^2$	X_{sj}
(s,t)	$E[X_{si}] + X_{it}$
(j,t)	X_{jt}

(b) Arc Travel Time Random Variables.

Figure 4-2: The Wheatstone Bridge (Figure 3-1) after one iteration of Algorithm 4.4. Figure 4-2(a) displays the resulting graph and Table 4-2(b) lists the corresponding travel time random variables.

Although the correctness of this algorithm generally follows from Garman's work, there is one difference in our implementation. As we noted in Section 4.2.1, we short-circuit the parallel arc removal when $|N^+(i)| \geq 2$ (or $|N^-(i)| \geq 2$). This effectively changes the order of parallel reductions from Garman's original algorithm. However, it does not change the resulting minimum travel time distribution P^*_{st}. To see this, consider a node i with two outgoing parallel arcs, $(i,j)^1, (i,j)^2$. If these parallel arcs are removed before **Step 2**, and an arc (l,i) is selected as a Type-1 arc in **Step 2**, Algorithm 4.4 will add a new arc (l,j) with $X_{lj} = E[X_{li}] + \mathcal{MIN}(X^1_{ij}, X^2_{ij})$. If these parallel arcs are not removed before **Step 2**, and an arc (l,i) is selected as a Type-1 arc in **Step 2**, Algorithm 4.4 will add two new arcs, $(l,j)^1, (l,j)^2$ with $X^1_{lj} = E[X_{li}] + X^1_{ij}$ and $X^2_{lj} = E[X_{li}] + X^2_{ij}$. When arcs $(l,j)^1, (l,j)^2$ are removed later with a parallel reduction, the resulting arc (l,j) will have $X_{lj} = \mathcal{MIN}(E[X_{li}] + X^1_{ij}, E[X_{li}] + X^2_{ij})$. We know that $E[X_{li}] + \mathcal{MIN}(X^1_{ij}, X^2_{ij}) = \mathcal{MIN}(E[X_{li}] + X^1_{ij}, E[X_{li}] + X^2_{ij})$ (see Section 2.4.2). Therefore X_{li} is the same even if we short-circuit the parallel arc removal in **Step 1**. A similar argument holds for Type-2 arcs.

In the case of the Wheatstone Bridge in Figure 3-1, Algorithm 4.4 identifies arc (s,i) as a Type-1 arc, and removes it to obtain the graph in Figure 4-2(a). In this new graph, the arc travel time random variables have been modified to the values in Table 4-2(b). After conditioning on arc (s,i), the graph in Figure 4-2(a) is now series-parallel reducible. In the next iteration, P^*_{st} will be calculated and the algorithm will terminate.

As with Algorithm 4.3, we have the following result for Algorithm 4.4:

Theorem 5. *Let P_{st} represent the travel time of a path π_{st} from source s to sink t in a non-series-parallel graph G'. Then $E[P^*_{st}] \leq E[P_{st}] \forall \pi_{st} \in \Pi_{st}$ where P^*_{st} is calculated by Algorithm 4.4.*

Proof. Algorithm 4.4 is composed of two functions: the sum and the minimum of random variables, and the operation of conditioning a Type-1 or Type-2 arc on its expected value. π_{st} is necessarily composed of arcs that will be identified as arcs in series, arcs in parallel, Type-1 arcs, or Type-2 arcs. Therefore, to show $E[P^*_{st}] \leq E[P_{st}]$, it is enough to show 1) $E[X_{ik}] \leq E[X_{ij}] + E[X_{jk}]$ for two arcs $(i,j),(j,k)$ in series and 2) $E[X_{ij}] \leq E[\mathcal{MIN}(X^1_{ij}, X^2_{ij})]$ for arcs $(i,j)^1, (i,j)^2$ in parallel 3) $E[X_{lk}] \leq E[X_{li}] + E[X_{ik}], (i,k) \in A^+(i)$ for all Type-1 arcs (l,i) and 4) $E[X_{kj}] \leq E[X_{ki}] + E[X_{ij}], (k,i) \in A^-(i)$ for all Type-2 arcs (i,j).

In the proof of Theorem 4, we show 1) and 2). To show 3), we note that for a Type-1 arc (l,i) and $\forall (i,k) \in A^+(i)$, we replace arcs $(l,i),(i,k)$ with a new arc (l,k), and we set $X_{lk} = E[X_{li}] + X_{ik}$. Therefore, $E[X_{lk}] = E[X_{li}] + E[X_{ik}]$. A similar approach for Type-2 arcs can be used to show 4). □

Note that with a small modification of Algorithm 4.4 we can create upper and lower bounding distributions for the minimum travel time distribution. Let UB_{li} be an upper-bound for the possible realizations of a Type-1 arc (l,i) and let UB_{ij} be an upper-bound for the possible realizations of a Type-2 arc (i,j). If we modify Algorithm 4.4 so that $X_{lk} = UB_{li} + X_{ik}$ for all Type-1 arcs (l,i), and $X_{lj} = X_{li} + UB_{ij}$ for all Type-2 arcs (i,j), then the resulting distribution P^*_{st} is an upper-bounding distribution for the minimum travel time distribution. Similarly, let LB_{li} be an lower-bound for the possible realizations of a Type-1 arc (l,i) and let LB_{ij} be an lower-bound for the possible realizations of a Type-2 arc (i,j). If we modify Algorithm 4.4 so that $X_{lk} = LB_{li} + X_{ik}$ for all Type-1 arcs (l,i), and $X_{lj} = X_{li} + LB_{ij}$ for all Type-2 arcs (i,j), then the resulting distribution P^*_{st} is a lower-bounding distribution for the minimum travel time distribution.

Chapter 5

Practical Implementation and Numerical Issues

In this chapter, we address two issues related to the implementation of Algorithm 2.3. First, we discuss methods to generate an acyclic network from a possibly cyclic network. This is necessary to use our routing procedure in practical scenarios. Second, we consider the efficient numerical implementation of the sum and the minimum functions. Efficient implementations of these functions are necessary for the calculation of minimum travel time distributions on large networks.

5.1 Two-Terminal Directed Acyclic Graph Generation

The minimum travel time distribution algorithms presented in Chapter 4 address **Step 2** of Algorithm 2.3. One of the requirements of these algorithms is that the input graph G' is a $\mathcal{TT}$-$\mathcal{DAG}$.

In most practical applications, such a $\mathcal{TT}$-$\mathcal{DAG}$ G' needs to be generated from a directed graph G, which is the network model of the application. This is **Step 1** of Algorithm 2.3. In each execution of Algorithm 2.3, a $\mathcal{TT}$-$\mathcal{DAG}$ G' needs to be generated for each $j \in N^+(i)$. However, depending on the method used to generate G', it may be possible to generate these graphs *a priori*. This is the approach we take in this thesis.

In [15], it is observed that generating such a $\mathcal{TT}$-$\mathcal{DAG}$ in probabilistic networks is not a trivial task. A natural approach to generating G' is to remove arcs from G until no cycles exist in G. Intuitively, such a *graph reduction* would remove all arcs that have no probability of being used to reach the sink t. However, determining the probability that an arc (i, j) is used to reach a sink t is not obvious, nor it is clear what an optimal acyclic graph generation entails, or even if such a graph generation exists.

Given these observations and a source s and sink t, our goal is to generate a $\mathcal{TT}$-$\mathcal{DAG}$ G' from G where G' is composed of all arcs that have some potential of being used to travel from source s to sink t. We use the informal term *potential* to

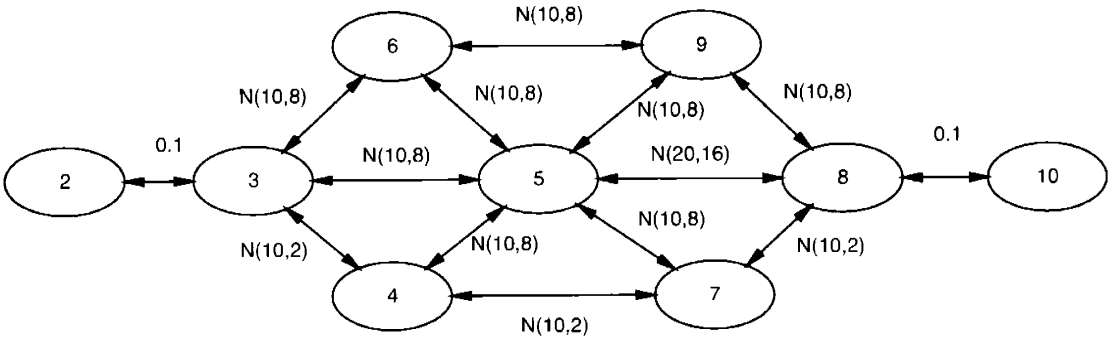


Figure 5-1: An Example Network.

emphasize that we do not formally calculate a *probability* that an arc will be used. As suggested in Section 3.5, one approach is to take some union of the k shortest simple paths from source s to sink t. We do not use this approach for two reasons. First, it is not clear how to appropriately select k. Second, taking the union of simple paths does not guarantee the resulting graph will by acyclic.

Instead, our approach is to perform the generation via an algorithm similar to Dial's (see Section 3.5). In an analogous fashion to Dial, we present our goals for such an algorithm:

1. The algorithm should include all arcs (i, j) that have a potential to be used to travel from source s to sink t. This implies there should be no cycles in G' (see the assumption in Section 2.3.3).

2. The algorithm should attempt to include as many arcs as possible in G'. Since it is difficult to say which arcs have no potential to be used, we attempt to include as many arcs as possible.

3. Consider a $\mathcal{TT}$-$\mathcal{DAG}$ G' from source s to sink t with $i \in N'(G' = (N', A'))$. An application of the algorithm from node i to sink t should generate a graph G'' such that G'' is a subgraph of G'.

4. The algorithm should avoid path enumeration.

5. The algorithm should run efficiently.

If the graphs generated by an algorithm with these goals become too complex, an additional goal for the algorithm could be to generate series-parallel graphs or graph components. In this chapter, we discuss four different algorithms and we illustrate the algorithms on the graph in Figure 5-1.

5.1.1 Simple Paths

For a given source s and a sink t, a simple approach to generating a $\mathcal{TT}$-$\mathcal{DAG}$ G' is to fix the arc travel times to $x_{ij} = E[X_{ij}]$ and run a one-to-one fastest path algorithm

from source s to sink t. G' can then be constructed by adding all arcs in the fastest path from source s to sink t. One of the primary benefits of this approach is that G' can be generated rapidly. Another benefit is that P^*_{st} can be calculated exactly and efficiently since G' is composed of arcs in series from source s to sink t. At the same time, the primary problem with this approach is that, since G' is only composed of a simple path, it excludes other arcs that might have some potential of being used in the fastest path. We do not illustrate this algorithm on Figure 5-1 since, for a given source s and sink t, G' can be determined by inspection.

5.1.2 Dial's Efficient Paths

In Section 3.5, we discuss Dial's algorithm for the assignment of traffic to multiple (s,t)-paths. One key aspect of the algorithm is the definition of an efficient path. Dial defines an efficient path as a path where every arc (i,j) has node i closer to source s than node j and node j closer to sink t than node i. For this thesis, the most important aspect of Dial's algorithm is that it generates a $\mathcal{TT}$-$\mathcal{DAG}$ subgraph G' of G where each arc in G' is part of at least one efficient path from source s to sink t. We do not consider the final step of Dial's algorithm (the assignment of traffic to arcs).

Algorithm 5.1 presents Dial's algorithm as we apply it to probabilistic networks. In this case, we redefine an efficient (s,t)-path as a path where, for every arc (i,j), the expected travel time from source s to node i is less than the expected travel time from source s to node j, and the expected travel time from node j to sink t is less than the expected travel time from node i to sink t. In this thesis, we discuss some of the problems and ambiguities associated with the use of expected travel times. However, for the algorithms in this chapter, the use of expectation is not as problematic since we only use expectation to generate a $\mathcal{TT}$-$\mathcal{DAG}$ G' from which we calculate P^*_{st}. In other words, we only use expected travel time to determine a subgraph G' of G whose arcs have some potential of being used.

Initially, all arc travel times are set to their expected values. Using these values, an all-to-one fastest path algorithm is executed in **Step 2** to compute a fastest path tree oriented toward sink t. Let $E[D_{it}]$ be the expected travel time from node i to sink t. In **Step 3**, a one-to-all fastest path algorithm is executed to compute a fastest path tree from source s. Let $E[D_{si}]$ be the expected travel time from source s to node i. In **Step 3**, the likelihoods of each arc are calculated. When an arc meets our definition of being efficient, we assign it a likelihood of 1. This is suggested in [7] for the case when we are only interested in efficient paths (and not traffic assignment).

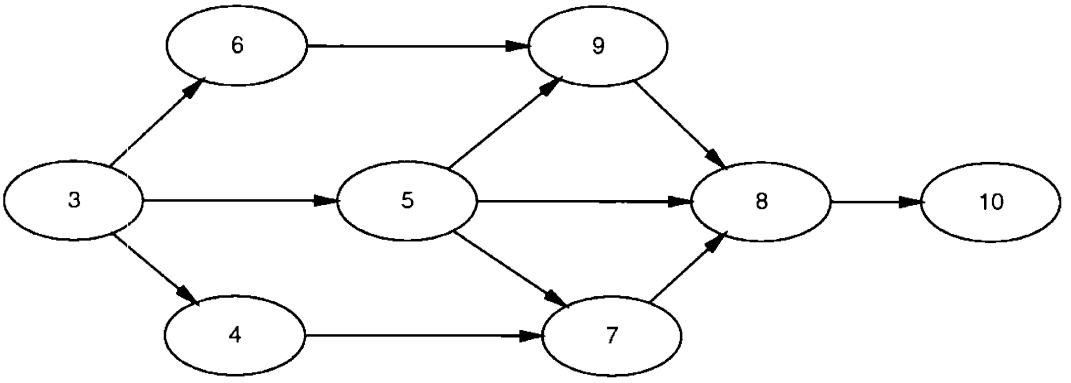
In the final step, the arc weights are calculated. Arcs with non-zero weights are added to G'. In the implementation, Q is a queue that contains the nodes of G in increasing order of the expected travel time from source s. At each iteration, a node i is taken from the front of Q and each arc $(i,j) \in A^+(i)$ is considered. If $i = s$, $w(i,j)$ is set to the likelihood $L(i,j)$. If $i \neq s$, $w(i,j)$ is calculated iteratively according to Dial's algorithm (see Section 3.5). Finally, if $w(i,j) > 0$ then arc (i,j) is added to G'.

We observe that Algorithm 5.1 performs a finite number of steps since we only

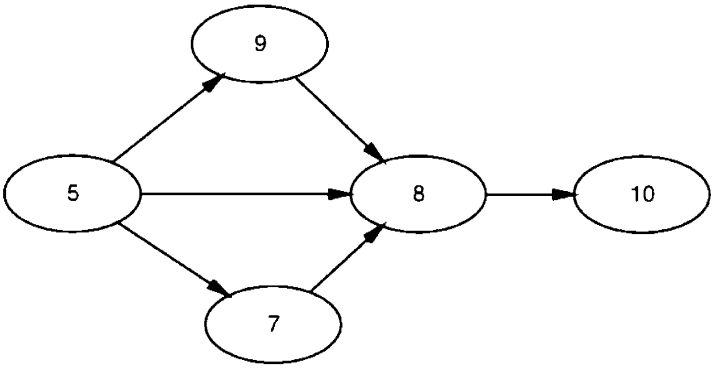
Input: $G = (N, A)$. $s, t \in N$.
Output: $G' = (N', A')$. G' is a $\mathcal{TT}$-$\mathcal{DAG}$ from s to t.

Step 1: Initialization.
for all $(i, j) \in A$ **do**
  $x_{ij} \leftarrow E[X_{ij}]$
Step 2: Run an all-to-one fastest path algorithm to t using x_{ij}.
Step 3: Run a one-to-all fastest path algorithm from s using x_{ij}.
Step 4: Calculate the likelihoods.
for $(i, j) \in A$ **do**
  if $E[D_{si}] < E[D_{sj}]$ AND $E[D_{jt}] < E[D_{it}]$ **then**
    $L(i, j) \leftarrow 1$
  else
    $L(i, j) \leftarrow 0$
Step 5: Calculate the weights and create G'.
$N' \leftarrow \{\}$
$A' \leftarrow \{\}$
$Q \leftarrow$ Sort $i \in N$ by increasing values of $E[D_{si}]$
while $Q.length() \geq 1$ **do**
  $i \leftarrow Q.pop\_front()$
  for $(i, j) \in A^+(i)$ **do**
    if $i \neq s$ **then**
      $w(i, j) \leftarrow 0$
      for $(u, i) \in A^-(i)$ **do**
        $w(i, j) \leftarrow w(i, j) + w(u, i)$
      $w(i, j) \leftarrow L(i, j) \cdot w(i, j)$
    else
      $w(i, j) \leftarrow L(i, j)$
    if $w(i, j) > 0$ **then**
      $N' \leftarrow N' \cup i$
      $N' \leftarrow N' \cup j$
      $A' \leftarrow A' \cup (i, j)$

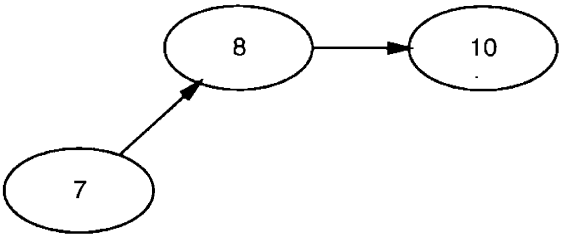
Algorithm 5.1: Dial's Algorithm as Applied to Probabilistic Networks.



(a) Subgraph Generated from Node 3 to Node 10.



(b) Subgraph Generated from Node 5 to Node 10.



(c) Subgraph Generated from Node 7 to Node 10.

Figure 5-2: The Subgraphs Generated by Algorithm 5.1 on Figure 5-1 for a User Located at Node 4 destined for Node 10.

remove nodes from Q. The running time of Algorithm 5.1 is dominated by the complexity of the fastest path algorithms. If $\overline{FP}$ is the complexity of the fastest path algorithm that is used, then the complexity of Algorithm 5.1 is $O(\overline{FP})$.

For completeness, we verify that G' is a $\mathcal{TT}$-$\mathcal{DAG}$ from source s to sink t.

Lemma 4 (Directed Acyclic Graphs with Algorithm 5.1). *Algorithm 5.1 generates a $\mathcal{TT}$-$\mathcal{DAG}$ from source s to sink t.*

Proof. To show this, we need to show that G' is acyclic and that it only has a single source s and sink t. By the specification of Algorithm 5.1, every arc in G' will be efficient such that $E[D_{si}] < E[D_{sj}]$ and $E[D_{jt}] < E[D_{it}]$. Therefore, every path in G' is also efficient and, by definition, cannot contain a cycle. No arc (i, s) will ever be added to G' since $E[D_{si}] > E[D_{ss}] = 0$. Similarly, no arc (t, j) will ever be added to G' since $E[D_{jt}] > E[D_{tt}] = 0$. Therefore, $|N^-(s)| = 0$ and $|N^+(t)| = 0$. □

To illustrate the generation of G' with Algorithm 5.1, we consider the network depicted in Figure 5-1. Suppose a user is located at node 4 with a final destination of node 10. According to Algorithm 2.3, we need to generate three directed acyclic graphs; one for each $j \in N^+(i)$. Therefore, we run Algorithm 5.1 three times with source-sink pairs $(3, 10)$, $(5, 10)$, and $(7, 10)$. In practice, we would naturally omit the adjacent node j where the user just came from. The resulting graphs after executing this algorithm are depicted in Figure 5-2(a) (node 3), Figure 5-2(b) (node 5), and Figure 5-2(c) (node 7).

We can modify the definition of Algorithm 5.1 in two ways. First, Dial discusses the removal of the condition that $E[D_{jt}] < E[D_{it}]$. This produces a larger graph G' and there may be more (s, t)-paths, but we can no longer guarantee that G' will be a $\mathcal{TT}$-$\mathcal{DAG}$ from source s to sink t. G' will still be acyclic, but there may be multiple sources and sinks. Therefore, we would also need to change Algorithm 5.1 to recursively remove any extra sources and sinks.

Another modification might be to allow arcs with $E[D_{si}] \leq E[D_{sj}]$ and $E[D_{jt}] \leq E[D_{it}]$. This would also produce a larger graph G' but we can no longer guarantee G' is acyclic (or that there will be a single source and sink). We would need to modify Algorithm 5.1 to account for these problems.

5.1.3 Fastest Paths Tree

It is hard to say which arcs have no potential in being used on the fastest path from source s to sink t. In a deterministic network, Dial attempts to address this by introducing the notion of an efficient path. In the previous section, we alter Dial's definition so it applies to probabilistic networks.

As stated, one of our goals in the construction of G' is to include as many arcs as possible. The requirements for an arc to be included in G' by Algorithm 5.1 are relatively strict. For comparative purposes, it is useful to consider other, more permissive, methods for the generation of a $\mathcal{TT}$-$\mathcal{DAG}$ in a probabilistic network.

It seems like it would make sense to include every node $i \in N$ in a simple path from source s to sink t. Intuitively, this would ensure a large number of paths and,

consequently, include a large number of arcs in G'. If we set the (positive) arc travel times to the expected travel times as before, it also seems like this can be accomplished by taking the union of the fastest path tree from source s with the fastest path tree to sink t. In this case, we would have a path from source s to every node $i \in N$, and a path from every node $i \in N$ to sink t. However, just taking the union of these two trees does not guarantee the resulting graph will be a $\mathcal{TT}$-$\mathcal{DAG}$. Furthermore, the arcs selected would still have to meet relatively strict requirements.

Instead, we proceed as follows. Rather than calculate the one-to-all fastest paths from source s, we only calculate the all-to-one fastest paths to sink t. This ensures that every node $i \in N$ has a path to sink t. G' is initialized to include the fastest paths tree to t. We want to add as many additional arcs to G' provided that G' remains acyclic. In particular, since it is difficult to determine which arcs have no potential, we explicitly ignore the characteristics of the travel time on any additional arcs. We also need G' to have a single source and sink.

One way to achieve this is to assign a topological order to the nodes in G' using the fastest paths tree to sink t. Once we have a topological ordering of the nodes, we can add all arcs that obey the topological ordering and still guarantee G' is acyclic. We ensure G' has a single source by recursively removing all other sources.

The algorithm is presented in Algorithm 5.2. Like Algorithm 5.1, Algorithm 5.2 is initialized by setting all arc travel times to the corresponding expected arc travel times. This is followed by the generation of the fastest path tree to sink t using an all-to-one fastest path algorithm. Let $succ(i)$ be the successor node of node i in the expected travel time fastest path tree oriented toward sink t ($succ(i)$ is closer to sink t than node i). We assign an order to each node based on a "breadth-first-walk" of the fastest path tree to sink t. Using this topological ordering, we add all arcs (i, j) such that $order(i) < order(j)$, where $order(i)$ is the topological order of node i. We know that there can only be one sink in G' since, by the construction of the fastest path tree to sink t, all nodes $i \neq t$ must have $|N^+(i)| > 0$. We recursively remove all sources by removing all nodes $i \neq s$ with $|N^-(i)| = 0$. Q is a generic queue data structure.

As before, we consider the network depicted in Figure 5-1 with a user at node 4 destined for node 10. According to Algorithm 2.3, we need to run Algorithm 5.2 three times with source-sink pairs $(3, 10)$, $(5, 10)$, and $(7, 10)$. The resulting graphs for node 3, node 5, and node 7 are presented in Figure 5-3(a), Figure 5-3(b), and Figure 5-3(c) respectively.

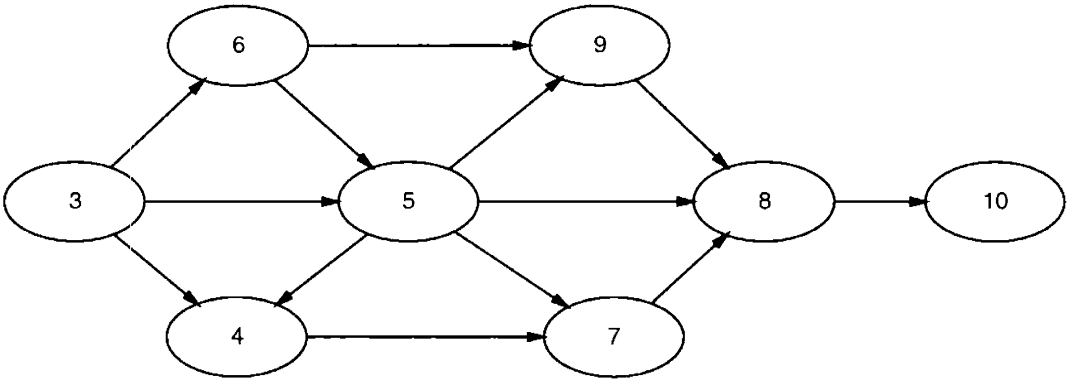
Algorithm 5.2 generally includes more arcs than Algorithm 5.1. However, the key part of Algorithm 5.2 is the topological ordering. The topological ordering is somewhat arbitrary, and different topological orderings can have an impact on the structure and size of G'.

We observe that since every step of Algorithm 5.2 is finite, the algorithm must also be finite. The running time of this algorithm is again dominated by the running time of the fastest path algorithm. If $\overline{FP}$ is the complexity of the fastest path algorithm that is used, then the complexity of Algorithm 5.2 is $O(\overline{FP})$. However, we note that a useful property of this algorithm is that the specification of the source node s is only necessary for **Step 6**. Therefore, if the results of **Step 1-Step 5** are cached, G'

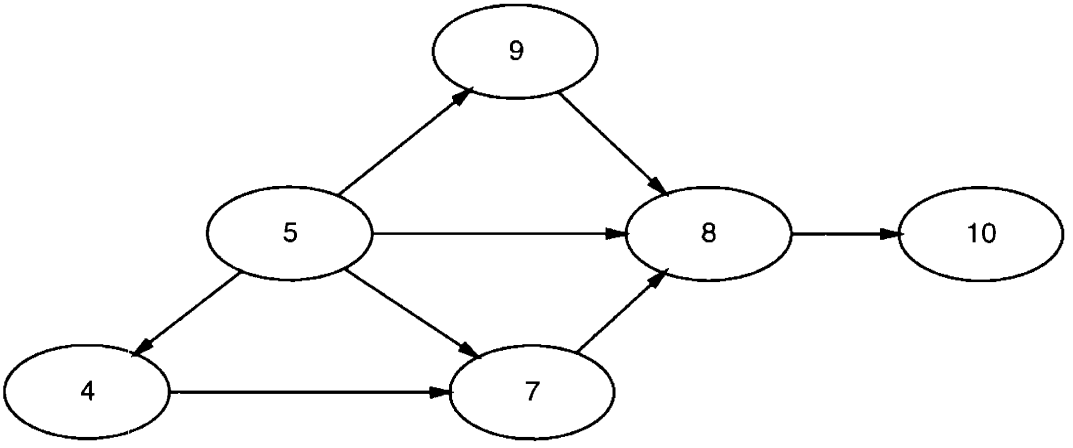
Input: $G = (N, A)$. $s, t \in N$.
Output: $G' = (N', A')$. G' is a $\mathcal{TT}$-$\mathcal{DAG}$ from s to t.

Step 1: Initialization.
for all $(i, j) \in A$ **do**
  $x_{ij} \leftarrow E[X_{ij}]$
Step 2: Run an all-to-one shortest path algorithm to t using x_{ij}.
$N' \leftarrow N$
$A' \leftarrow \{\}$
Step 3: Add all arcs in the fastest path tree to t to $G' = (N', A')$.
for all $i \in N$ **do**
  $j \leftarrow succ(i)$
  $A' \leftarrow A' \cup (i, j)$
Step 4: Create a topological ordering for $G' = (N', A')$.
$Q.enqueue(t)$
$order\_number \leftarrow |N'|$
$order(t) = order\_number$
while $Q.size() \geq 1$ **do**
  $i \leftarrow Q.pop\_front()$
  for all $j \in N^-(i)$ (in G') **do**
    $order\_number \leftarrow order\_number - 1$
    $order(j) = order\_number$
    $Q.enqueue(j)$
Step 5: Add any arcs from G that satisfy the topological order.
for all $(i, j) \in A$ **do**
  if $order(i) < order(j)$ **then**
    $A' \leftarrow A' \cup (i, j)$
Step 6: Recursively remove all nodes in G' with $|N^-(i)| = 0$, $i \neq s$.
$Q.clear()$
for all $i \in N' \setminus s$ **do**
  if $|N^-(i)| = 0$ **then**
    $Q.enqueue(i)$
while $Q.size() \geq 1$ **do**
  $i \leftarrow Q.pop\_front()$
  for all $j \in N^+(i)$ **do**
    for $k \in 2 \ldots |A(i, j)|$ **do**
      $A' \leftarrow A' \setminus (i, j)^k$
    if $j \neq s$ AND $|N^-(j)| = 0$ **then**
      $Q.enqueue(j)$
  $N' \leftarrow N' \setminus i$

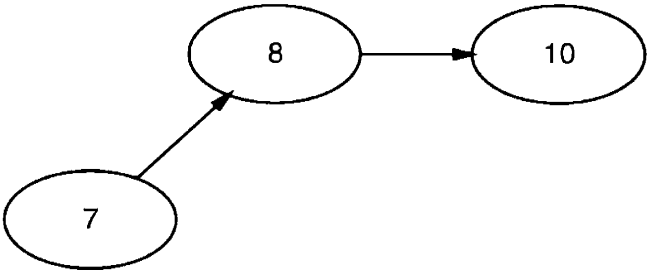
Algorithm 5.2: Fastest Paths Tree.



(a) Subgraph Generated from Node 3 to Node 10.



(b) Subgraph Generated from Node 5 to Node 10.



(c) Subgraph Generated from Node 7 to Node 10.

Figure 5-3: The Subgraphs Generated by Algorithm 5.2 on Figure 5-1 for a User Located at Node 4 destined for Node 10.

can be calculated in linear time for every post-initial execution of Algorithm 5.2.

For completeness, we also verify that Algorithm 5.2 does produce a $\mathcal{TT}$-$\mathcal{DAG}$ G' from source s to sink t.

Lemma 5 (Directed Acyclic Graphs with Algorithm 5.2). *Algorithm 5.2 generates a $\mathcal{TT}$-$\mathcal{DAG}$ from source s to sink t.*

Proof. A directed graph G' is acyclic if and only if there exists a topological ordering of G'. Initially, Algorithm 5.2 creates a tree to sink t. Any tree is acyclic and, thus, has a topological ordering. By only adding arcs to the tree that obey the topological ordering, we cannot create a cycle. Therefore, the graph G' generated by Algorithm 5.2 must be acyclic. To see the graph has two terminals, it suffices to note that all nodes i with $|N^-(i)| = 0$ are recursively removed from G'. By the construction of the fastest paths tree, the only node i with $|N^+(i)| = 0$ is the sink t. □

5.1.4 More on Efficient Paths

The arcs included by Algorithm 5.1 need to meet relatively strict requirements. Algorithm 5.2 is more permissive and attempts to include more arcs. The main problem with Algorithm 5.2 is that the structure of the resulting $\mathcal{TT}$-$\mathcal{DAG}$ G' depends on the precise topological ordering. In this section, we consider another approach. The intent of this third approach is to generate a $\mathcal{TT}$-$\mathcal{DAG}$ G' that is larger than the $\mathcal{TT}$-$\mathcal{DAG}$ produced by Algorithm 5.1. We also want to avoid the arbitrary nature of the topological ordering in Algorithm 5.2.

In [7], Dial discusses the removal of $E[D_{jt}] < E[D_{it}]$ in the definition of an efficient path. In a similar fashion, we consider the removal of the condition, $E[D_{si}] < E[D_{sj}]$. In other words, an arc (i, j) will only be declared efficient if node j is closer to the sink t than node i in expected travel time. In general, more arcs will be efficient because this criteria represents a relaxation of the previous efficient path definition. If we use Algorithm 5.1 with this new criteria, we cannot guarantee the resulting graph will be a $\mathcal{TT}$-$\mathcal{DAG}$ since it is possible to have multiple sources. However, by adding arcs in an appropriate manner, we can "grow" a $\mathcal{TT}$-$\mathcal{DAG}$ from source s to sink t.

In addition to including arcs with $E[D_{jt}] < E[D_{jt}]$, consider the special case of adding the arc (i, j) when $E[D_{jt}] = E[D_{it}]$. Algorithm 5.1 will not add arc (i, j) to G', but should it? $E[D_{it}] = E[D_{jt}]$ implies that sometimes the actual travel time from node i to sink t will be better than the actual travel time from node j to sink t, and other times the actual travel time from node i to sink t will be worse than the actual travel time from node j to sink t. In other words, it seems that arc (i, j) has some potential to be used in a fastest path from source s to sink t. We note that, in practice, this special case of $E[D_{it}] = E[D_{jt}]$ is unlikely to occur. However, we include it here because $E[D_{jt}] \leq E[D_{it}]$ is a relaxed condition compared to $E[D_{jt}] < E[D_{it}]$ and because, even though less likely, $E[D_{jt}] = E[D_{it}]$ still can occur.

Using this intuition, we formally redefine an efficient path to be a path where each arc (i, j) in the path satisfies the criteria $E[D_{jt}] \leq E[D_{it}]$. There are, however, some subtleties in this definition. The same logic that we use to justify the addition of arc

(i, j) when $E[D_{jt}] = E[D_{it}]$, also justifies the addition of arc (j, i) (if it exists in G). If both arc (i, j) and arc (j, i) are added to G', then G' will not be a $\mathcal{TT}$-$\mathcal{DAG}$ since G' will contain a cycle. Therefore, we need to further differentiate between arc (i, j) and arc (j, i). At the same time, we observe that this differentiation might become quite complex as we will have to compare more and more characteristics of the arcs. Furthermore, depending on the level of detail and type of comparison, there is still some chance that arc (i, j) will be considered an equal of arc (j, i). In such a case, we resort to a arbitrary selection of one of these arcs.

In this third approach to generating a $\mathcal{TT}$-$\mathcal{DAG}$, we perform the differentiation by calculating an additional metric $H(i)$ for each node i. For a given sink t, $H(i)$ is the number of arcs from node i to sink t. Using $H(i)$, we redefine an efficient path. An arc (i, j) is efficient if $E[D_{jt}] < E[D_{it}]$ or $E[D_{jt}] = E[D_{it}]$ with $H(j) < H(i)$. While this latter condition is still somewhat arbitrary, we use it for three reasons:

- Given two paths with equal expected travel times, in some situations we can make the case that a user will prefer the path with fewer arcs. For example, in a transportation network, a path with fewer arcs (roads) will have fewer nodes (intersections). Though we do not explicitly model delays at nodes, fewer intersections means fewer influences on travel time (such as delays related to merging).

- The purpose of defining efficient paths is to generate a "reasonable" $\mathcal{TT}$-$\mathcal{DAG}$ G'. G' is one component of a larger routing procedure. We should not make the generation of G' overly complex or time-consuming. With random travel times, there is an almost infinite number of ways to compare arc (i, j) with arc (j, i). A useful property of $H(i)$ is that we can determine $H(i) \, \forall \, i \in N$ in linear time with a Breadth-First-Search (BFS).

- $H(i)$ is only used in a special case. In practice, it is unlikely that $E[D_{jt}]$ will be exactly equal to $E[D_{it}]$.

If $E[D_{jt}] = E[D_{it}]$ with $H(j) = H(i)$, then we arbitrarily judge arc (i, j) and arc (j, i) by an ordering of the node names. For example, if the node names are numeric with $i = 4, j = 5$ and $E[D_{jt}] = E[D_{it}]$ with $H(j) = H(i)$, then we add arc $(4, 5)$ rather than arc $(5, 4)$.

Given these observations, Algorithm 5.3 implements our redefinition of an efficient path. Though it is similar to Algorithm 5.1, Algorithm 5.3 "grows" a $\mathcal{TT}$-$\mathcal{DAG}$ from source s and does not calculate arc likelihoods $L(i, j)$. In the first step, $E[D_{it}]$ is calculated for each $i \in N$. Similarly, in **Step 2**, $H(i)$ is calculated for each $i \in N$. Based on our redefinition of an efficient path, there are three ways to add an arc (i, j) to G'. First, we include arcs (i, j) with $E[D_{jt}] < E[D_{it}]$ (node j is closer in expected travel time to sink t than node i). Second, in the special case of $E[D_{jt}] = E[D_{it}]$, we include arc (i, j) if $H(j) < H(i)$. Third, if $E[D_{jt}] = E[D_{it}]$ and $H(j) = H(i)$, we arbitrarily declare arc (i, j) to be efficient if the node name of i precedes the node name of j in some type of ordering. In the implementation in Chapter 6, the node

names are numeric so we will include arc (i,j) if $E[D_{jt}] = E[D_{it}], H(j) = H(i)$ and $i < j$.

We add arcs to G' by starting at source s and "growing" G'. With each iteration, the first node i in the queue Q is removed and all arcs (i,j) with $(i,j) \in A^+(i)$ are considered for addition to G'. If an arc (i,j) meets any of the three criteria for being considered efficient, (i,j) is added to G'. Node j is only added to Q if it has not been placed in Q before. This is accomplished by setting the marked flag for each node. Note that since Q is a queue, G' is constructed in a breadth-first manner.

Since each node i is added to Q only once and the calculation of $H(i)$ takes $O(m)$ time with a BFS, the complexity of this Algorithm 5.3 is dominated by the complexity of the fastest path algorithm. Therefore, Algorithm 5.3 runs in $O(\overline{FP})$.

We also ensure that Algorithm 5.3 produces a $\mathcal{TT}$-$\mathcal{DAG}$ from source s to sink t.

Lemma 6 (Directed Acyclic Graphs with Algorithm 5.3). *Algorithm 5.3 generates a $\mathcal{TT}$-$\mathcal{DAG}$ from source s to sink t.*

Proof. Suppose a cycle $\hat{C}$ is created when arc (i,j) is added to G'. Then there exists a path π_{ji} from node j to node i. There are three ways to add arc (i,j) to G' with Algorithm 5.3.

In the first case, suppose arc (i,j) is added to G' because $E[D_{jt}] < E[D_{it}]$. Each arc $(k,l) \in \pi_{ji}$ has the property that $E[D_{lt}] \leq E[D_{kt}]$, which means that $E[D_{it}] \leq E[D_{jt}]$. But this is not possible since we assumed arc (i,j) was added because $E[D_{jt}] < E[D_{it}]$.

In the second case, suppose arc (i,j) is added because $E[D_{jt}] = E[D_{it}]$ and $H(j) < H(i)$. It must be the case that each arc $(k,l) \in \pi_{ji}$ has the property that $E[D_{lt}] = E[D_{kt}]$. It must also be the case that each arc $(k,l) \in \pi_{ji}$ has the property that $H(l) \leq H(k)$. This means that $H(i) \leq H(j)$, but this is a contradiction since we assumed arc (i,j) was added because $E[D_{jt}] = E[D_{it}]$ and $H(j) < H(i)$.

In the third case, suppose arc (i,j) is added because $E[D_{jt}] = E[D_{it}], H(j) = H(i)$ and $i < j$. It must be the case that each arc $(k,l) \in \pi_{ji}$ has the two properties $E[D_{lt}] = E[D_{kt}]$ and $H(k) = H(l)$. It must also be the case that each arc $(k,l) \in \pi_{ji}$ has the property that $k < l$. This means that $j < i$, but this is a contradiction since we assumed arc (i,j) was added because $E[D_{jt}] = E[D_{it}], H(j) = H(i)$ and $i < j$.

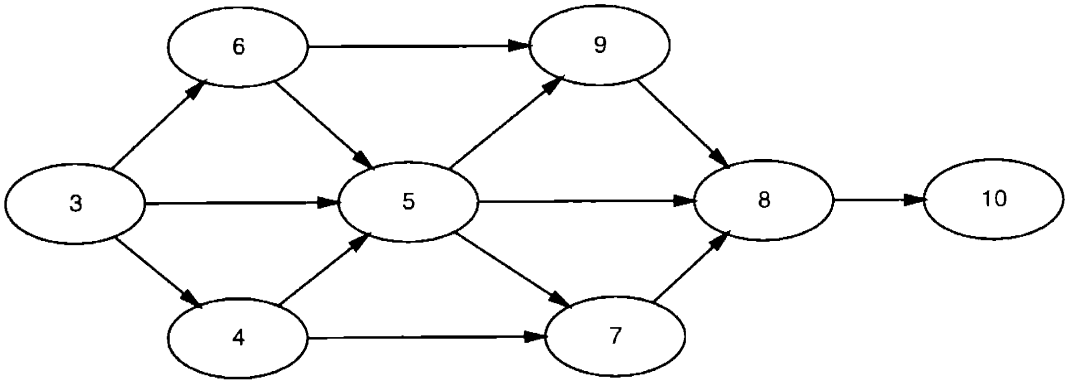
Since all three cases have contradictions, our assumption that a cycle $\hat{C}$ exists is incorrect and G' must be acyclic. To see there are only two terminals, it is enough to observe that G' is constructed from source s to sink t. □

As before, we consider the network depicted in Figure 5-1 with a user at node 4 destined for node 10. According to Algorithm 2.3, we need to run Algorithm 5.3 three times with source-sink pairs $(3,10)$, $(5,10)$, and $(7,10)$. The resulting graphs for node 3, node 5, and node 7 are presented in Figure 5-4(a), Figure 5-4(b), and Figure 5-4(c) respectively.

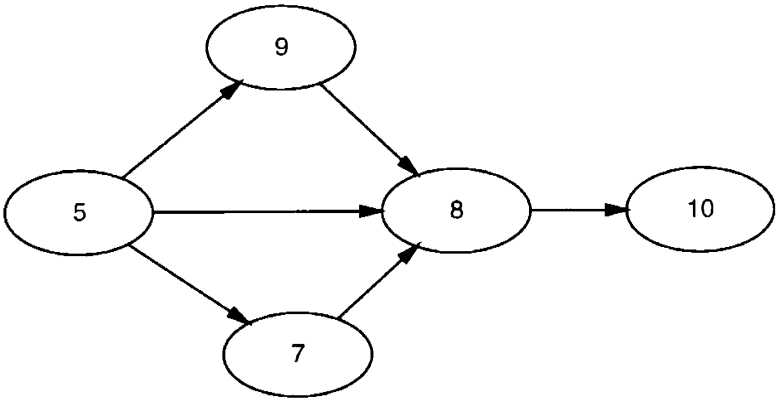
Input: $G = (N, A)$. $s, t \in N$.
Output: $G' = (N', A')$. G' is a $\mathcal{TT}$-$\mathcal{DAG}$ from s to t.

```
for all (i, j) ∈ A do
    x_ij ← E[X_ij]
Step 1: Run an all-to-one fastest path algorithm to t using x_ij.
for all (i, j) ∈ A do
    x_ij ← 1
Step 2: Run an all-to-one fastest path algorithm to t using x_ij.
N' ← {}
A' ← {}
for all i ∈ N do
    marked(i) = FALSE
Step 3: Add all (i, j) ∈ A+(i) where arc (i, j) is efficient.
Q.enqueue(s)
marked(s) = TRUE
while Q.length() ≥ 1 do
    i ← Q.pop_front()
    for (i, j) ∈ A+(i) do
        if E[D_jt] < E[D_it] then
            N' ← N' ∪ i
            N' ← N' ∪ j
            A' ← A' ∪ (i, j)
            if marked(j) = FALSE then
                Q.enqueue(j)
                marked(j) = TRUE
        else if E[D_jt] = E[D_it] AND H(j) < H(i) then
            N' ← N' ∪ i
            N' ← N' ∪ j
            A' ← A' ∪ (i, j)
            if marked(j) = FALSE then
                Q.enqueue(j)
                marked(j) = TRUE
        else if E[D_jt] = E[D_it] AND H(j) = H(i) AND i < j then
            N' ← N' ∪ i
            N' ← N' ∪ j
            A' ← A' ∪ (i, j)
            if marked(j) = FALSE then
                Q.enqueue(j)
                marked(j) = TRUE
```

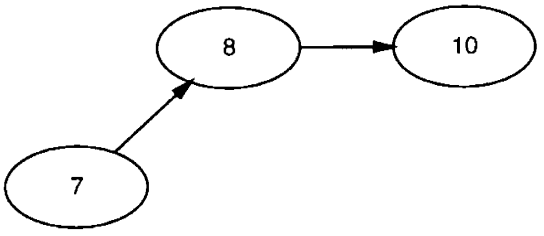
Algorithm 5.3: More on Efficient Paths.



(a) Subgraph Generated from Node 3 to Node 10.



(b) Subgraph Generated from Node 5 to Node 10.



(c) Subgraph Generated from Node 7 to Node 10.

Figure 5-4: The Subgraphs Generated by Algorithm 5.3 on Figure 5-1 for a User Located at Node 4 destined for Node 10.

5.2 Numerical Routines

The second problem this chapter addresses is the implementation of numerical routines to perform the sum and the minimum of random variables. The importance of efficient implementations cannot be overstated: these routines will be repeatedly executed and, aside from our assumptions, cannot assume any special characteristics of the random variables. Before these routines can be addressed, we need to consider how to discretely represent continuous random variables. Recall that we model arc travel times as continuous random variables.

5.2.1 Random Variable Representation

It is not clear how to represent random variables such that functions of random variables can be repeatedly performed in a time and space efficient manner. Interestingly, in the research reviewed, Martin [26] appears to be the only one to explicitly consider this question. His approach is to approximate probability density functions with polynomial functions. This is similar to the PERT/CPM approach of using the beta distribution with optimistic, pessimistic, and average value parameters.

There are three issues to address in the representation of random variables:

- We must decide what aspect of the random variable to characterize. For example, we can characterize a random variable as a probability density function, as a cumulative distribution function, or as various statistics. Naturally, using the probability density function or the cumulative distribution function allows us to completely represent a random variable. In this thesis, we use the probability density function.

- Second, once we have decided on which aspect to characterize, we must decide how to represent it. For example, we can represent a probability density function with Martin's polynomial representation, a more general polynomial representation, or symbolically. A fourth method is to use a pair of arrays. In this case, one array consists of the possible realizations of the random variable, while the other array consists of the corresponding densities. The actual probability density function can then be reconstructed with interpolation.

- Third, we cannot develop a representation without an understanding of several numerical algorithms. This is because the representation determines which numerical algorithms can be used and, conversely, the arguments to the numerical algorithms determine how the random variables can be represented. For example, if each random variable is characterized by its probability density function, we will need to perform an integration to obtain the cumulative distribution function. Similarly, if each random variable is characterized as a cumulative distribution function, we will need differentiation to get the probability density function. Of course, we could avoid integration and differentiation altogether by storing both the density function and distribution function. Numerical integration and differentiation routines typically take functions as arguments.

A final consideration is that our methods should be able to be used on-line. In such a situation, rather than knowing the arc travel time distributions *a priori*, we might want to estimate the arc travel time distributions with values that are observed during the course of operation. By definition, probability density estimators take an array of sampled values as input.

Given these observations, our solution is to use an *approximate random variable* to discretely represent a continuous random variable. An approximate random variable with ρ possible realizations is composed of an ordered array $\mathbf{x} = [x_1\ x_2\ \ldots\ x_\rho]$, $x_1 < x_2 < \ldots < x_\rho$, $x_i \in \mathbb{R}^+\ \forall i \in \{1, \ldots, \rho\}$, and the array $\mathbf{y} = [y_1\ y_2\ \ldots\ y_\rho]$, $y_i \in \mathbb{R}^+ \cup \{0\}\ \forall i \in \{1, \ldots, \rho\}$. Furthermore, for $i \in \{1, \ldots, \rho - 1\}$, we assume that the distance between any two consecutive elements x_i, x_{i+1} in $\mathbf{x}$ is the same.

In this representation, $\mathbf{x}$ is an array of possible realizations of the random variable and $\mathbf{y}$ is an array of the corresponding densities. Note that we do not assume that we necessarily know the density function $f_X(x)$; we just have the outputs $\mathbf{y} \approx f_X(\mathbf{x})$. We can construct $f_X(\mathbf{x})$ by *interpolating* over $\mathbf{x}$ and $\mathbf{y}$. In other words, for a random variable X, we represent X as:

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & \ldots & x_\rho \\ y_1 & y_2 & \ldots & y_\rho \end{bmatrix} \approx \begin{bmatrix} \mathbf{x} \\ f_X(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & \ldots & x_\rho \\ f_X(x_1) & f_X(x_2) & \ldots & f_X(x_\rho) \end{bmatrix}$$

For the complete implementation, we propose the data structure in Table 5.1. In addition to the two arrays mentioned, the data structure also records the finite minimum and maximum values for $\mathbf{x}$ and $\mathbf{y}$ (denoted $\min(\mathbf{x}), \max(\mathbf{x}), \min(\mathbf{y}), \max(\mathbf{y})$). Under the assumptions in Chapter 2, we have $a = \min(\mathbf{x}), b = \max(\mathbf{x})$. This means we assume $y_i = 0\ \forall x_i \notin [\min(\mathbf{x}), \max(\mathbf{x})]$. Strictly speaking, it is not necessary to record $\min(\mathbf{x})$ and $\max(\mathbf{x})$ since $\mathbf{x}$ is ordered and, therefore, $\min(\mathbf{x}) = x_1$ and $\max(\mathbf{x}) = x_\rho$. However, for clarity in presentation, we include $\min(\mathbf{x})$ and $\max(\mathbf{x})$ in the data structure.

We also define the scalar $\delta(\mathbf{x}) = \frac{\max(\mathbf{x}) - \min(\mathbf{x})}{\rho - 1}$ as the *interval width*. Since we assume that the distance between any two consecutive elements x_i, x_{i+1} in $\mathbf{x}$ is the same, $\delta(\mathbf{x})$ is $x_{i+1} - x_i\ \forall i \in \{1, \ldots, \rho - 1\}$. Intuitively, $\delta(\mathbf{x})$ represents how "finely" a random variable is represented. In our implementation, the number of possible realizations ρ of a random variable depends on how the random variable is created. For each arc travel time random variable X_{ij} that is defined as part of the network model, ρ is assumed to be given. In this case, the choice of ρ depends on a variety of factors including the interpolation routine that is used, the desired accuracy of the approximation of $f_{X_{ij}}(x_{ij})$, and the computational resources that are available. The other way we create a random variable is as the result of the sum or the minimum of random variables. In this case, ρ is determined by the number of possible realizations of each argument to the sum and minimum routines. Note that this data structure applies to univariate arc travel time random variables. These ideas extend naturally to multivariate random variables.

For the sum of random variables, this array-based approach allows for the use of several fast convolution algorithms which are primarily taken from the field of

Type	Data	Comment
array	$\mathbf{x}$	The array of possible realizations
array	$\mathbf{y}$	The array of densities
scalar	$\min(\mathbf{x})$	The finite minimum of all possible realizations
scalar	$\min(\mathbf{y})$	The finite minimum of all possible densities
scalar	$\max(\mathbf{x})$	The finite maximum of all possible realizations
scalar	$\max(\mathbf{x})$	The finite maximum of all possible densities
scalar	$\delta(\mathbf{x})$	The interval width

Table 5.1: Data Structure for a Univariate Random Variable X.

signal processing [29]. Two efficient algorithms include direct convolution and Fast Fourier Transform-based convolution. The minimum of random variables is not as well-researched as convolution, and we resort to an efficient calculation based on the definition of the minimum. With two random variables, the resulting equation involves the multiplication of the density and distribution functions (see Section 2.4.2). Numerical integration and interpolation are two fundamental routines that are repeatedly used in our implementation of the sum and minimum functions.

5.2.2 The Numerical Sum of Random Variables - Convolution

We consider two approaches to convolution using the data structure in Table 5.1. The first approach uses a linear time-invariant (LTI) filter to implement a *direct convolution*. In the second approach, we take the characteristic function of each random variable and use Fast Fourier Transforms. A third approach would be to use the definition of the sum of random variables and convolution explicitly. However, we do not pursue this approach as the LTI filter approach is very similar in terms of complexity, and faster in practice.

Random Variable Scaling

Using the data structure in Table 5.1, we say two random variables A, B are on the same *scale* if

$$\begin{aligned} \delta(\mathbf{x}^{(A)}) &= \delta(\mathbf{x}^{(B)}) \\ \max(\mathbf{x}^{(A)}) &= \max(\mathbf{x}^{(B)}) \\ \min(\mathbf{x}^{(A)}) &= \min(\mathbf{x}^{(B)}) \end{aligned}$$

where the superscript $(\cdot)$ denotes the random variable that the data structure refers to. In other words, for the random variables A and B to be on the same scale, then they have the same range of realizations with the same interval width. Note there is no reference to the corresponding density values $\mathbf{y}^{(A)}, \mathbf{y}^{(B)}$ in our definition

of scale. This means that if A and B are on the same scale, it is not necessarily the case that $y_i^{(A)}$ equals $y_i^{(B)}$. By setting A and B to the same scale before we convolve the corresponding density functions, we can simplify the operations required for convolution.

Algorithm 5.4 is the scaling routine that is run before convolution. First, it calculates δ, the interval width of each random variable after being scaled. In Algorithm 5.4, $\delta(\mathbf{x}^{(A')})$ and $\delta(\mathbf{x}^{(B')})$ are each set to the larger of $\delta(\mathbf{x}^{(A)})$ and $\delta(\mathbf{x}^{(B)})$. Next, the range of realizations for each random variable is expanded and reset to be the same. LB is the lower-bound on this range, and UB is the upper-bound on this range. Finally, for both A and B, we recalculate the arrays of densities $\mathbf{y}^{(A)}$ and $\mathbf{y}^{(B)}$ by interpolating over the new set of possible realizations.

In terms of complexity, the major operation in Algorithm 5.4 is interpolation. The complexity of the interpolation depends on the interpolation routine that is used. In this thesis, we use cubic spline interpolation since we assume the arc travel times are continuous and the distance between any two consecutive realizations x_i, x_{i+1} in $\mathbf{x}$ ($i \in \{1, \ldots, \rho - 1\}$) is the same. With cubic spline interpolation the array of realizations $\mathbf{x}$ is equivalent to the array of interpolation "knots". Cubic spline interpolation requires an initialization which takes $O(\rho)$ to solve a tridiagonal system of equations [33]. The lookup of an interpolated value involves a bisection search on a table generated by the initialization [33]. In the worst case, there are $\rho = \frac{UB-LB}{\delta(\mathbf{x})} + 1$ bisection searches. Each bisection search takes $O(log_2\rho)$ so the total complexity of this routine is $O(\rho \log_2 \rho)$. However, since x_i' is incremented sequentially in Algorithm 5.4, the search can sometimes be achieved in linear time using an interpolation routine that caches previous lookups.

Direct Convolution

Direct convolution is common in several procedures from digital signal processing. In the end, it is similar to computing a convolution by definition, but more efficient in practice. If we want to filter the array $\mathbf{z}$ by an array $\frac{\mathbf{u}}{\mathbf{v}}$ (element-wise division), then we can solve the following LTI difference equation to get the resulting array $\mathbf{w}$ [29]:

$$\sum_{k=1}^{|\mathbf{v}|} v_k w_{j+1-k} = \sum_{i=1}^{|\mathbf{u}|} u_i z_{j+1-i}$$

where we denote the number of elements in the array $\mathbf{v}$ ($\mathbf{u}$) as $|\mathbf{v}|$ ($|\mathbf{u}|$). Note that if $|\mathbf{v}| = 1$ and $v_1 = 1$, this equation reduces to:

$$w_j = \sum_{i=1}^{|\mathbf{u}|} u_i z_{j+1-i} \tag{5.1}$$

Recall that the convolution equation derived for the density function of $C = A + B$

Input: A, B are random variables.
Output: A', B' are random variables with the same interval width and range of realizations.

Step 1: Calculate the interval width δ.
if $\delta(\mathbf{x}^{(A)}) > \delta(\mathbf{x}^{(B)})$ **then**
  $\delta \leftarrow \delta(\mathbf{x}^{(A)})$
else
  $\delta \leftarrow \delta(\mathbf{x}^{(B)})$
Step 2: Set the range of realizations to be the same for A and B.
if $\min(\mathbf{x}^{(A)}) < \min(\mathbf{x}^{(B)})$ **then**
  $LB \leftarrow \min(\mathbf{x}^{(A)})$
else
  $LB \leftarrow \min(\mathbf{x}^{(B)})$
if $\max(\mathbf{x}^{(A)}) < \max(\mathbf{x}^{(B)})$ **then**
  $UB \leftarrow \max(\mathbf{x}^{(B)})$
else
  $UB \leftarrow \max(\mathbf{x}^{(A)})$
Step 3: Recalculate the densities $\mathbf{y}$ for the rescaled A and B.
$\delta(\mathbf{x}^{(A')}) \leftarrow \delta$
$\delta(\mathbf{x}^{(B')}) \leftarrow \delta$
$i \leftarrow 1$
$x'_i \leftarrow LB$
while $x'_i \leq UB$ **do**
  if $x'_i < \min(\mathbf{x}^{(A)})$ **then**
    $y_i^{(A')} \leftarrow 0$
  else if $x'_i > \max(\mathbf{x}^{(A)})$ **then**
    $y_i^{(A')} \leftarrow 0$
  else
    $y_i^{(A')} \leftarrow interpolate(x'_i, \mathbf{x}^{(A)}, \mathbf{y}^{(A)})$
  $x_i^{(A')} \leftarrow x'_i$
  if $x'_i < \min(\mathbf{x}^{(B)})$ **then**
    $y_i^{(B')} \leftarrow 0$
  else if $x'_i > \max(\mathbf{x}^{(B)})$ **then**
    $y_i^{(B')} \leftarrow 0$
  else
    $y_i^{(B')} \leftarrow interpolate(x'_i, \mathbf{x}^{(B)}, \mathbf{y}^{(B)})$
  $x_i^{(B')} \leftarrow x'_i$
  $x'_{i+1} \leftarrow x'_i + \delta$
  $i \leftarrow i + 1$

Algorithm 5.4: Random Variable Scaling.

is:

$$f_C(c) = \int_{\min(\mathbf{x}^{(A)})}^{\max(\mathbf{x}^{(A)})} f_A(a) f_B(c-a)\, da \tag{5.2}$$

If we discretize this equation for implementation using the trapezoid approximation, we have:

$$f_C(c) \approx \delta \cdot \sum_{\min(\mathbf{x}^{(A)})}^{\max(\mathbf{x}^{(A)})} f_A(a) f_B(c-a) \tag{5.3}$$

where the random variables A and B are on the same scale and δ is the interval width. To see the correspondence between convolution and the LTI difference equation, we rewrite Equation 5.3 using our random variable data structure:

$$y_j^{(C)} \approx \delta \cdot \sum_{i=1}^{\rho} y_i^{(A)} y_{j+1-i}^{(B)}$$

where ρ is the number of possible realizations for A and B. Therefore, if we take $y_j^{(C)} = w_j, y_i^{(A)} = u_i, y_{j+1-i}^{(B)} = z_{j+1-i}$ then Equation 5.1 is exactly the same as Equation 5.3 without δ. Using a Direct Form II Transposed "filter" [29], we can solve Equation 5.1 for $\mathbf{w}$ in $O(|\mathbf{u}|^2)$, which means we can solve for $\mathbf{y}^{(C)}$ in $O(\rho^2)$. Note that although this is theoretically equivalent to calculating a convolution by definition, it is faster in practice since there is less overhead in the Direct Form II implementation.

The complete convolution routine is presented in Algorithm 5.5. Initially, we put the random variables A and B on the same scale. With A' and B' on the same scale with ρ' possible realizations, there are $2\rho' - 1$ possible realizations for $C = A + B$[1]. Therefore, we need to "right-hand-side" pad A' and B' with enough zeros so that $|\mathbf{x}^{(A')}| = 2\rho' - 1, |\mathbf{x}^{(B')}| = 2\rho' - 1$. Next, the filter is used to convolve the two densities. We normalize the output with an element-wise multiplication of $\mathbf{y}^{(C)}$ by $\delta(\mathbf{x}^{(C)})$. In the next step, we need to set the range of possible realizations. Since this is a convolution, the minimum possible realization of C is $\min(\mathbf{x}^{(A')}) + \min(\mathbf{x}^{(B')})$ and the maximum possible realization is $\max(\mathbf{x}^{(A')}) + \max(\mathbf{x}^{(B')})$.

As noted, the complexity of this operation is $O(\rho^2)$ where ρ is the number of realizations of each random variable after being scaled. In this thesis, ρ is a user-defined parameter. If ρ is very small, then the interval width $\delta(\mathbf{x}) = \frac{\max(\mathbf{x}) - \min(\mathbf{x})}{\rho - 1}$ is large. Intuitively, this means the realizations are well-separated, which could lead to erroneous interpolations. At the same time, since there would be fewer realizations, the convolution would run faster.

[1]Let A and B be on the same scale. If we add a single possible realization of A to each possible realization of B, there are ρ' possible realizations for C. Now if we add a second possible realization of A to each possible realization of B, there is only one new possible realization for C. Continuing on, we see there are $2\rho' - 1$ possible realizations for C.

Input: A and B are random variables.
Output: $C \leftarrow A + B$.
Step 1: Put A and B on the same scale.
$A', B' \leftarrow$ Scale A, B with Algorithm 5.4
Step 2: Pad A and B with zeros.
$\rho' \leftarrow \frac{\max(\mathbf{x}^{(A')}) - \min(\mathbf{x}^{(A')})}{\delta(\mathbf{x}^{(A')})} + 1$ {Note: The choice of A' is arbitrary}
$i \leftarrow \rho' + 1$
while $i \leq 2\rho' - 1$ **do**
  $x_i^{(A')} \leftarrow 0$
  $x_i^{(B')} \leftarrow 0$
  $i \leftarrow i + 1$
Step 3: Perform the convolution.
$\delta(\mathbf{x}^{(C)}) \leftarrow \delta(\mathbf{x}^{(A')})$ {Note: The choice of A' is arbitrary}
$\mathbf{y}^{(C)} \leftarrow filter(\mathbf{y}^{(A')}, \mathbf{y}^{(B')})$
$\mathbf{y}^{(C)} \leftarrow \mathbf{y}^{(C)} \cdot \delta(\mathbf{x}^{(C)})$ {Note: Element-wise normalization}
Step 4: Set the range of possible realizations for C.
$\min(\mathbf{x}^{(C)}) \leftarrow \min(\mathbf{x}^{(A')}) + \min(\mathbf{x}^{(B')})$
$\max(\mathbf{x}^{(C)}) \leftarrow \max(\mathbf{x}^{(A')}) + \max(\mathbf{x}^{(B')})$
$i \leftarrow 1$
$x'_i \leftarrow \min(\mathbf{x}^{(C)})$
while $x'_i \leq \max(\mathbf{x}^{(C)})$ **do**
  $x_i^{(C)} \leftarrow x'_i$
  $x'_{i+1} \leftarrow x'_i + \delta(\mathbf{x}^{(C)})$
  $i \leftarrow i + 1$

Algorithm 5.5: The Sum of Random Variables with Direct Convolution.

Fast Fourier Transform Convolution

Though direct convolution is relatively efficient with a $O(\rho^2)$ complexity, a Fast Fourier Transform (FFT) convolution runs in $O(\rho \log_2 \rho)$. To get a feel for the size of this improvement, consider the sum of two random variables, each with 1000 possible realizations ($\rho = 1000$). A direct convolution would take on the order of 1000000 operations, while an FFT-based convolution would take on the order of 10000 operations, which is a significant improvement in run-time.

Given a probability density function $f_X(x)$, its characteristic function, χ, is defined as:

$$\chi(f_X(x)) = \int_{-\infty}^{\infty} e^{itx} f_X(x)\, dx$$

which is a particular case of the Fourier Transform of $f_X(x)$ (denoted $\mathcal{FFT}(f_X(x))$):

$$\mathcal{FFT}(f_X(x)) = \sqrt{\frac{|b|}{(2\pi)^{1-a}}} \int_{-\infty}^{\infty} e^{ibtx} f_X(x)\, dx$$

with $a = 1$ and $b = 1$ (see [31] and a summary in [46]). From an implementation standpoint, we naturally use the discrete Fourier Transform. The fundamental observation for an FFT-based convolution is the *Convolution Theorem* which says:

$$\begin{aligned}
C &= A + B \\
f_C(c) &= f_A(a) * f_B(b) \\
\mathcal{FFT}(f_C(c)) &= \mathcal{FFT}(f_A(a)) \cdot \mathcal{FFT}(f_B(b)) \text{ Element-wise multiplication} \\
f_C(c) &= \mathcal{FFT}^{-1}(\mathcal{FFT}(f_A(a)) \cdot \mathcal{FFT}(f_B(b))) \text{ Element-wise multiplication}
\end{aligned}$$

where $\mathcal{FFT}^{-1}$ is the inverse Fourier Transform. The Convolution Theorem says we can calculate $f_C(c)$ by taking the inverse Fourier Transform of the element-wise product of the Fourier Transforms of $f_A(a)$ and $f_B(b)$. Calculating these Fourier Transforms can be accomplished in $O(\rho \log_2 \rho)$ time with the FFT [33].

Algorithm 5.6 presents an FFT-based convolution routine. Like Algorithm 5.5, we first put the random variables A and B on the same scale. The padding procedure is also the same, except that we pad out to the largest power of two greater than $2\rho' - 1$, which is denoted ρ^*. This is done so the FFT can operate without any additional memory overhead. The convolution is calculated according to the Convolution Theorem, except we take the backwards FFT rather than the inverse FFT. The backwards FFT is an unscaled inverse FFT. Therefore, we need to normalize the result so $f_C(c) \approx \mathbf{y}^{(C)}$ is a proper density function. This is done by dividing $\mathbf{y}^{(C)}$ by the normalization constant $K = \int_{\min(\mathbf{x}^{(C)})}^{\max(\mathbf{x}^{(C)})} f_C(c)\, dc$. K can be evaluated by a combined interpolation and numerical integration routine. If the density functions are known not to have any values greater than 1, we can replace the calculation of K with ρ^*.

Input: A and B are random variables.
Output: $C \leftarrow A + B$.
Step 1: Put A and B on the same scale.
$A', B' \leftarrow$ Scale A, B with Algorithm 5.4
Step 2: Pad A and B with zeros up to a power of two.
$\rho' \leftarrow \frac{\max(\mathbf{x}^{(A')}) - \min(\mathbf{x}^{(A')})}{\delta(\mathbf{x}^{(A')})} + 1$ {Note: The choice of A' is arbitrary}
$\rho^* \leftarrow 2$
while $\rho^* \leq (2\rho' - 1)$ **do**
  $\rho^* \leftarrow (\rho^*)^2$
$i \leftarrow \rho' + 1$
while $i \leq \rho^*$ **do**
  $x_i^{(A')} \leftarrow 0$
  $x_i^{(B')} \leftarrow 0$
  $i \leftarrow i + 1$
Step 3: Perform the convolution.
$\delta(\mathbf{x}^{(C)}) \leftarrow \delta(\mathbf{x}^{(A')})$ {Note: The choice of A' is arbitrary}
$\min(\mathbf{x}^{(C)}) \leftarrow \min(\mathbf{x}^{(A')}) + \min(\mathbf{x}^{(B')})$
$\max(\mathbf{x}^{(C)}) \leftarrow \max(\mathbf{x}^{(A')}) + \max(\mathbf{x}^{(B')})$
$\mathbf{y}^{(C)} \leftarrow \mathcal{FFT}^{-1}(\mathcal{FFT}(\mathbf{y}^{(A')}) \cdot \mathcal{FFT}(\mathbf{y}^{(B')}))$ {Note: Element-wise multiplication}
$K \leftarrow integrate(interpolate(\mathbf{x}^{(C)}, \mathbf{y}^{(C)}, \min(\mathbf{x}^{(C)}), \max(\mathbf{x}^{(C)})))$
$\mathbf{y}^{(C)}/K$ {Note: Element-wise normalization}
Step 4: Set the range of possible realizations for C.
$i \leftarrow 1$
$x'_i \leftarrow \min(\mathbf{x}^{(C)})$
while $x'_i \leq \max(\mathbf{x}^{(C)})$ **do**
  $x_i^{(C)} \leftarrow x'_i$
  $x'_{i+1} \leftarrow x'_i + \delta(\mathbf{x}^{(C)})$
  $i \leftarrow i + 1$

Algorithm 5.6: The Sum of Random Variables with Fast Fourier Transforms.

Controlling Convolution Run-time Let $\rho^{(C)}$ refer to the number of possible realizations of the random variable C, where C is the result of $A + B$. In both convolution routines, the consequence of scaling and padding A and B is that $\rho^{(C)}$ can become large, especially over a series of convolutions. More critically, after A and B are put on the same scale with ρ' realizations, the number of possible realizations for C is $\rho^{(C)} = 2\rho' - 1$. In a series of convolutions, $\rho^{(C)}$ determines the run-time of the next convolution, so with each iteration the convolution routine runs slower. One useful observation is that we can make $\rho^{(C)}$ smaller by removing realizations that have an extremely low probability of being realized. In fact, the zero-padding of A and B will create a relatively large number of realizations with this property.

We can also control large growth in $\rho^{(C)}$ by readjusting $\rho^{(C)}$ after C has been calculated. In Chapter 6, we compare two different methods for controlling $\rho^{(C)}$. In the first approach, we fix the interval width $\delta(\mathbf{x}^{(C)})$ to a constant value. Since $\delta(\mathbf{x}^{(C)}) = \frac{\max(\mathbf{x}^{(C)}) - \min(\mathbf{x}^{(C)})}{\rho - 1}$, the growth of $\rho^{(C)}$ is controlled by the range of possible realizations $(\max(\mathbf{x}^{(C)}) - \min(\mathbf{x}^{(C)}))$. In the second approach, we explicitly fix $\rho^{(C)}$ to a constant value. Since $\rho^{(C)} = \frac{\max(\mathbf{x}^{(C)}) - \min(\mathbf{x}^{(C)})}{\delta(\mathbf{x}^{(C)})} + 1$, the growth of $\delta(\mathbf{x}^{(C)})$ is controlled by the range of possible realizations $(\max(\mathbf{x}^{(C)}) - \min(\mathbf{x}^{(C)}))$. Generally speaking, as the density functions of more random variables are convolved (and the range of possible realizations increases), fixing $\rho^{(C)}$ to a constant value has the effect of increasing $\delta(\mathbf{x}^{(C)})$. The advantage of this approach is that every convolution will have a similar run-time.

We have suggested controlling $\rho^{(C)}$ as a means to control the run-time of the convolution routine. Note that in limiting $\rho^{(C)}$, we may lose some accuracy in the representation of the random variable C. The actual effects of limiting $\rho^{(C)}$ depend on the interpolation routine used in the convolution routine. In general, any attempt to control $\rho^{(C)}$ involves balancing the effects of $\rho^{(C)}$ on the run-time and the accuracy of the convolution routine. In Section 6.1, we discuss the practical impact of controlling $\rho^{(C)}$ in our convolution routines.

5.2.3 The Numerical Minimum of Random Variables

There is little prior work on the numerical calculation of the minimum of two random variables. As such, the approach in this thesis is to use the definition and Equation 2.2:

$$C = \mathcal{MIN}(A, B) \Leftrightarrow f_C(c) = f_A(c)[1 - F_B(c)] + f_B(c)[1 - F_A(c)]$$

The numerical procedure to calculate this equation is presented in Algorithm 5.7. In **Step 1**, the interval width and range of realizations for C are calculated to put A and B on the same scale. The minimum possible realization is the minimum of $\min(\mathbf{x}^{(A)})$ and $\min(\mathbf{x}^{(B)})$, while the maximum possible realization is the minimum of $\max(\mathbf{x}^{(A)})$ and $\max(\mathbf{x}^{(B)})$. In **Step 2**, we recalculate the arrays of densities $\mathbf{y}^{(A)}$ and $\mathbf{y}^{(B)}$ by interpolating over the new set of possible realizations. We also calculate the arrays $\mathbf{Y}^{(A)}$ and $\mathbf{Y}^{(B)}$. $\mathbf{Y}^{(A)}$ and $\mathbf{Y}^{(B)}$ correspond to arrays of the cumulative distributions of A and B respectively. The calculation of $\mathbf{Y}^{(A)}$ and $\mathbf{Y}^{(B)}$ requires

a combined interpolation and numerical integration routine. Once the values of $\mathbf{y}$ and $\mathbf{Y}$ are known for A and B, these arrays are multiplied together in accordance with Equation 2.2 to obtain $\mathbf{y}^{(C)}$. The complexity of this routine is $O(\rho^2)$ where $\rho = \frac{\max(\mathbf{x}^{(C)}) - \min(\mathbf{x}^{(C)})}{\delta(\mathbf{x}^{(C)})} + 1$. This is due to the $O(\rho)$ complexity of the combined interpolation and numerical integration routine.

Let $\rho^{(C)}$ refer to the number of possible realizations of C, where C is the result of $\mathcal{MIN}(A, B)$. Note that, unlike the convolution routines, after A and B are put on the same scale, our minimum routine does not further increase $\rho^{(C)}$. The practical impact of this property is that Algorithm 5.7 does not slow down as much as either Algorithm 5.5 or Algorithm 5.6 over a series of calls. Consequently, it is not necessary to consider methods to control $\rho^{(C)}$ in our minimum routine.

Input: A and B are random variables.
Output: $C = \mathcal{MIN}(A, B)$.

Step 1: Set the interval width and the range of possible realizations.
if $\delta(\mathbf{x}^{(A)}) > \delta(\mathbf{x}^{(B)})$ **then**
  $\delta(\mathbf{x}^{(C)}) \leftarrow \delta(\mathbf{x}^{(A)})$
else
  $\delta(\mathbf{x}^{(C)}) \leftarrow \delta(\mathbf{x}^{(B)})$
if $\min(\mathbf{x}^{(A)}) < \min(\mathbf{x}^{(B)})$ **then**
  $\min(\mathbf{x}^{(C)}) \leftarrow \min(\mathbf{x}^{(A)})$
else
  $\min(\mathbf{x}^{(C)}) \leftarrow \min(\mathbf{x}^{(B)})$
if $\max(\mathbf{x}^{(A)}) < \max(\mathbf{x}^{(B)})$ **then**
  $\max(\mathbf{x}^{(C)}) \leftarrow \max(\mathbf{x}^{(A)})$
else
  $\max(\mathbf{x}^{(C)}) \leftarrow \max(\mathbf{x}^{(B)})$
Step 2: Calculate the densities $\mathbf{y}$ and cumulative distributions $\mathbf{Y}$.
$i \leftarrow 1$
$x'_i \leftarrow \min(\mathbf{x}^{(C)})$
while $x'_i \leq \max(\mathbf{x}^{(C)})$ **do**
  if $x'_i < \min(\mathbf{x}^{(A)})$ **then**
    $Y_i^{(A)} \leftarrow 0$, $y_i^{(A)} \leftarrow 0$
  else if $x'_i > \max(\mathbf{x}^{(A)})$ **then**
    $Y_i^{(A)} \leftarrow 1$, $y_i^{(A)} \leftarrow 0$
  else
    $Y_i^{(A)} \leftarrow integrate(interpolate(\mathbf{x}^{(A)}, \mathbf{y}^{(A)}, \min(\mathbf{x}^{(A)}), x'_i))$
    $y_i^{(A)} \leftarrow interpolate(x'_i, \mathbf{x}^{(A)}, \mathbf{y}^{(A)})$
  if $x'_i < \min(\mathbf{x}^{(B)})$ **then**
    $Y_i^{(B)} \leftarrow 0$, $y_i^{(B)} \leftarrow 0$
  else if $x'_i > \max(\mathbf{x}^{(B)})$ **then**
    $Y_i^{(B)} \leftarrow 1$, $y_i^{(B)} \leftarrow 0$
  else
    $Y_i^{(B)} \leftarrow integrate(interpolate(\mathbf{x}^{(B)}, \mathbf{y}^{(B)}, \min(\mathbf{x}^{(B)}), x'_i))$
    $y_i^{(B)} \leftarrow interpolate(x'_i, \mathbf{x}^{(B)}, \mathbf{y}^{(B)})$
  $x'_{i+1} \leftarrow x'_i + \delta(\mathbf{x}^{(C)})$
  $i \leftarrow i + 1$
Step 3: Calculate $f_C(c) = f_A(c)[1 - F_B(c)] + f_B(c)[1 - F_A(c)]$ for all c.
$i \leftarrow 1$
$x_i^{(C)} \leftarrow \min(\mathbf{x}^{(C)})$
while $x_i^{(C)} \leq \max(\mathbf{x}^{(C)})$ **do**
  $y_i^{(C)} \leftarrow y_i^{(A)} \cdot (1 - Y_i^{(B)}) + y_i^{(B)} \cdot (1 - Y_i^{(A)})$
  $x_{i+1}^{(C)} \leftarrow x_i^{(C)} + \delta(\mathbf{x}^{(C)})$
  $i \leftarrow i + 1$

Algorithm 5.7: The Minimum of Random Variables.

Chapter 6

Results and Examples

Chapter 2 presents our approach to routing in probabilistic networks, and Chapters 4-5 discuss the key concepts in our approach. In this chapter, we illustrate how these ideas are actually applied to routing, and how we can improve the quality of routing in probabilistic networks. First, we discuss the performance of the numerical implementations of the sum and minimum functions. Efficient implementations of these routines are required for routing on large networks where we have to calculate the sum and the minimum of many random variables. We then illustrate the routing procedure on three networks. The first network is small and its structure is simple. The second network is small, but has a more complicated structure. Relative to these two small networks, the third network is much larger. This third network is a modified model of the Amsterdam A-10 beltway, and it is composed of 200 nodes and 522 arcs.

6.1 Performance of Numerical Routines

The efficiency of the sum and the minimum functions is the critical component in determining the run-time of the minimum travel time distribution algorithms described in Chapter 4. Examining the practical performance of these functions is necessary for understanding the effectiveness and limitations of our routing procedure. In addition to understanding the run-time of the direct convolution routine and the Fast Fourier Transform convolution routine, we also examine the effects of controlling ρ, which is the number of possible realizations of a random variable.

Our implementation of the sum and minimum functions used interpolation and numerical integration routines from a scientific library. All algorithms were implemented in C++ and were run with debugging and error-checking functions enabled, and without compiler optimizations. Therefore, the run-times reported should be used comparatively and not be considered as the fastest possible run-times. The machine used to obtain the results in this section was a 933 MHz Pentium 3 CPU with 256M of RAM, and the Linux 2.4 kernel.

6.1.1 Numerical Convolution

In Chapter 5 we describe two routines to calculate the density function of $C = A + B$. Let ρ be the number of possible realizations of A or B after A and B have been put on the same scale. A direct convolution implementation runs in $O(\rho^2)$ and a Fast Fourier Transform convolution implementation runs in $O(\rho \log_2 \rho)$.

Procedure

Both of our convolution routines (Algorithm 5.5 and Algorithm 5.6) take two random variables as arguments. For example, to sum five random variables we need to make four calls to a convolution routine. To understand the performance of Algorithm 5.5 and Algorithm 5.6, we numerically convolved the density functions of k independent $\mathcal{N}(10, 2)$ random variables with both routines, where k varied from 2 to 7. Each distribution had $\min(\mathbf{x}) = 5$ and $\max(\mathbf{x}) = 15$. Furthermore, for each random variable, the number of realizations ρ was set to a uniform random value between 800 and 1800.

Before discussing the convolution results, we consider the quality of the representation of the random variable A with our two array representation $\mathbf{x}^{(A)}, \mathbf{y}^{(A)}$. Let $\hat{E}[A]$ and $\widehat{Var}(A)$ be the values of $E[A]$ and $Var(A)$ under the two array representation. If A is distributed according to $\mathcal{N}(10, 2)$ with $\rho^{(A)} = 1024$, then our representation calculates $\hat{E}[A] = 9.99593$ and $\widehat{Var}(A) = 2.02897$. Furthermore, if we calculate the relative error as $\frac{|\hat{E}[A] - E[A]|}{E[A]}$ and $\frac{|\widehat{Var}(A) - Var(A)|}{Var(A)}$, we find $\frac{|\hat{E}[A] - E[A]|}{E[A]} = 0.000406$ and $\frac{|\widehat{Var}(A) - Var(A)|}{Var(A)} = 0.014487$. Expressed in percentages, this means the relative error in calculating the expectation is less than 0.1% and the relative error in calculating the variance is approximately 1.5%.

Convolution Run-time Base Case

An efficient implementation of a convolution routine is needed for routing on large networks where we have to sum many random variables.

Figure 6-1(a) compares the run-time of the convolution routines. Unlike the next two sections, in this base case we did not control ρ, the number of possible realizations. As is evident, the FFT-based convolution routine runs much faster than the direct convolution routine, especially as the number of the random variables in the sum increases (and ρ increases).

Table 6-1(b) compares the accuracy of the numerical convolution routines with the equivalent theoretically convolved density functions. Each row corresponds to the convolution of the density functions of a number of independent $\mathcal{N}(10, 2)$ distributions. For example, the row with $\mathcal{N}(30, 6)$ in the second column corresponds to the theoretical convolution of the density functions of three independent $\mathcal{N}(10, 2)$ random variables. For each of the two numerical convolution routines, there are three sub-columns in Table 6-1(b). The first sub-column is the relative error in expectation of the convolution routine, the second sub-column is the relative error in variance, and the third sub-column is the mean squared error (MSE). In our case, we define

MSE as $\frac{1}{\rho}\sum_{i=1}^{\rho}(f_{\mathcal{N}(E,Var)}(x_i) - y_i)^2$ where $f_{\mathcal{N}(E,Var)}$ is the relevant density function, and x_i, y_i are the ith elements of the two array representation of $\mathcal{N}(E, Var)$ obtained by numerical convolution. In other words, x_i is an element of the array of possible realizations of $\mathcal{N}(E, Var)$, and y_i is the corresponding density calculated by numerical convolution.

It is evident that both the direct convolution routine and the FFT-based routine produce accurate results. In both routines the relative error in expectation and variance is small, although the relative error in expectation is slightly smaller for the direct convolution routine. In terms of percentages, no relative error exceeds 0.55% for either convolution routine. The MSE is also small for both numerical convolution routines. Note that, in some cases, the values appears to be exactly equal between convolution routines. However, these values are rounded; the actual values differed slightly.

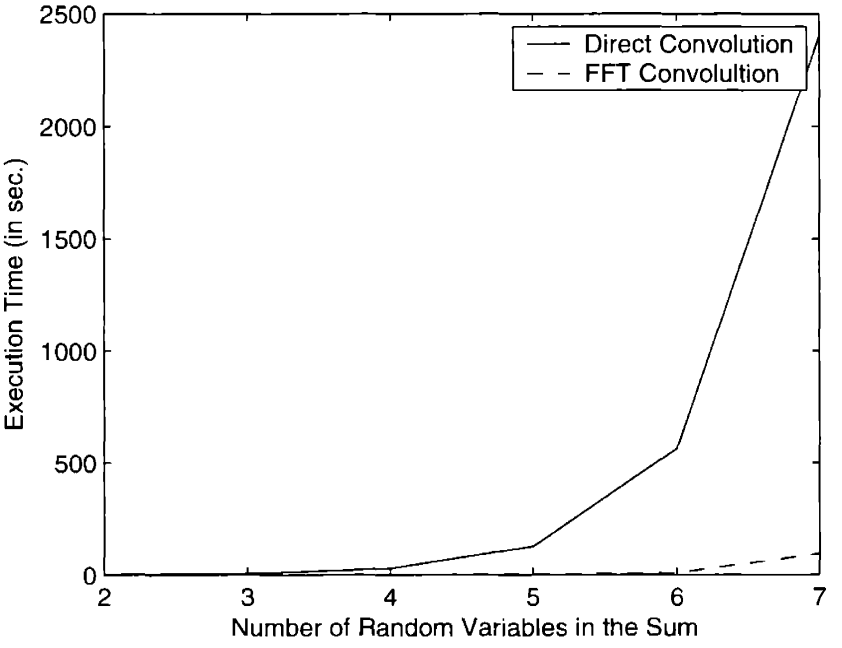
Controlling the Interval Width

In Section 5.2.2, we discuss two methods to control the run-time of a series of convolutions. As ρ is the principle factor in determining the run-time of both the direct convolution routine and the FFT-based convolution routine, both methods attempt to control the growth of the number of possible realizations ρ. The first method implicitly controls ρ by setting the interval width $\delta(\mathbf{x})$ to a constant value in **Step 1** of the random variable scaling procedure (Algorithm 5.4). Since $\delta(\mathbf{x}) = \frac{\max(\mathbf{x})-\min(\mathbf{x})}{\rho-1}$, the growth of ρ is controlled by the range of possible realizations $(\max(\mathbf{x}) - \min(\mathbf{x}))$.

Figure 6-2(a) compares the run-time of the two convolution routines. As expected, the FFT-based convolution routine runs much faster than the direct convolution routine, especially as the number of the random variables in the sum increases. In general, as the number of random variables in the sum increases, the range of possible realizations of the resulting random variable increases. This means ρ will also increase. If we compare Figure 6-2(a) to Figure 6-1(a), we see that this method of controlling ρ does improve the run-time when compared to the base case of not controlling ρ. However, this improvement only appears to be on the order of a constant improvement.

Table 6-2(b) compares the accuracy of the numerical convolution routines when we set the interval width $\delta(\mathbf{x})$ in the random variable scaling procedure (Algorithm 5.4). As before, each row corresponds to the convolution of the density functions of a number of independent $\mathcal{N}(10, 2)$ distributions. Similarly, for each of the two numerical convolution routines, there are three sub-columns in Table 6-2(b). The first sub-column is the relative error in expectation of the convolution routine, the second sub-column is the relative error in variance, and the third sub-column is the mean squared error (MSE). After some experimentation, $\delta(\mathbf{x})$ was set to $0.01955 = \frac{20}{1023}$.

As is the case in Table 6-1(b), it is evident that both the direct convolution routine and the FFT-based routine produce accurate results. In both routines, the relative error in expectation and variance is small, although the relative error in expectation is slightly smaller for the direct convolution routine. Note that for direct convolution, the relative error in expectation and variance is approximately constant. In terms of percentages, no relative error exceeds 0.54% for either convolution routine. The MSE



(a) Run-time.

	Type of Convolution						
	Theoretical	Direct			Fast Fourier Transform		
k	$\mathcal{N}(E, Var)$	$\frac{\lvert\hat{E}-E\rvert}{E}$	$\frac{\lvert\widehat{Var}-Var\rvert}{Var}$	MSE	$\frac{\lvert\hat{E}-E\rvert}{E}$	$\frac{\lvert\widehat{Var}-Var\rvert}{Var}$	MSE
2	$\mathcal{N}(20, 4)$	0.000001	0.005418	$< 10^{-8}$	0.000001	0.005418	$< 10^{-8}$
3	$\mathcal{N}(30, 6)$	$< 10^{-6}$	0.005406	$< 10^{-8}$	$< 10^{-6}$	0.005406	$< 10^{-8}$
4	$\mathcal{N}(40, 8)$	$< 10^{-6}$	0.005400	$< 10^{-8}$	$< 10^{-6}$	0.005400	$< 10^{-8}$
5	$\mathcal{N}(50, 10)$	$< 10^{-6}$	0.005397	$< 10^{-9}$	$< 10^{-6}$	0.005397	$< 10^{-9}$
6	$\mathcal{N}(60, 12)$	$< 10^{-6}$	0.005399	$< 10^{-9}$	0.000036	0.005318	$< 10^{-9}$
7	$\mathcal{N}(70, 14)$	$< 10^{-6}$	0.005393	$< 10^{-9}$	0.000049	0.005293	$< 10^{-9}$

(b) Error.

Figure 6-1: Base case performance of the numerical convolution routines. Figure 6-1(a) compares the run-time of the direct convolution routine with the FFT-based convolution routine. Table 6-1(b) compares the relative error in expectation and variance, and the MSE of the two convolution routines.

is also small for both numerical convolution routines.

Controlling the Number of Possible Realizations

The second method of controlling the run-time of the convolution routines is to explicitly set ρ. In general, as more random variables are summed, the range of possible realizations of the resulting random variable will increase. Therefore, setting ρ to a constant value has the effect of increasing the interval width $\delta(\mathbf{x})$. Intuitively, this means the random variables will become less "finely" discretized. The primary advantage of explicitly setting ρ is that every convolution will have a relatively similar run-time.

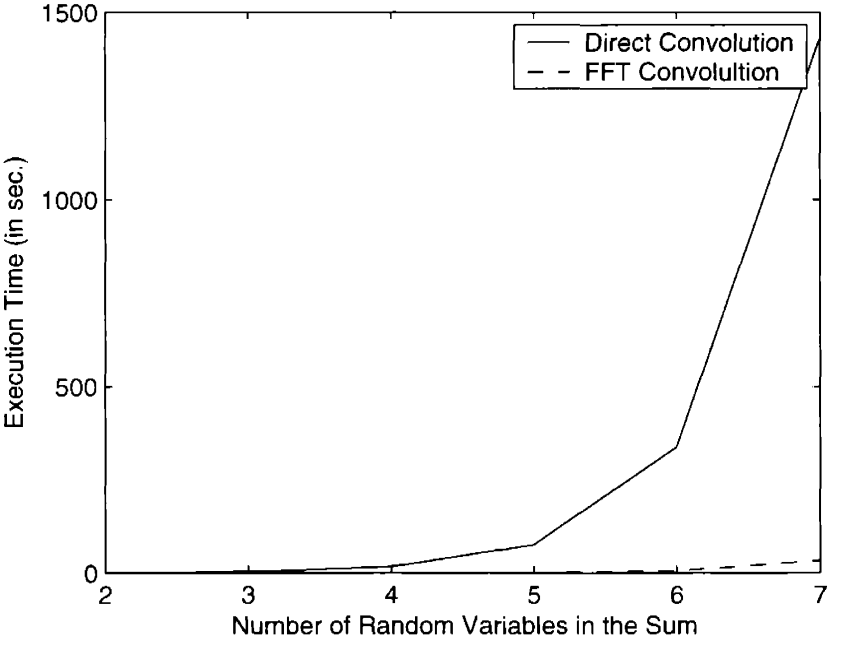
Figure 6-3(a) compares the run-time of the two convolution routines. As expected, the FFT-based convolution routine runs faster than the direct convolution routine. Unlike the previous two sections, both the direct convolution routine and FFT-based convolution routine have reasonable run-times. Note that the direct convolution run-time appears to increase linearly as the number of random variables in the sum increases. Although, from theory, we would expect the convolution run-times to be relatively constant, this small increase in run-time is due to the overhead of the scaling procedure (Algorithm 5.4). In any case, the direct convolution routine run-time is still quite fast. If we compare Figure 6-3(a) to Figure 6-1(a) and Figure 6-2(a), we see that this method of controlling ρ offers a large improvement in run-time.

Table 6-3(b) compares the accuracy of the numerical convolution routines when we explicitly set ρ in the random variable scaling procedure (Algorithm 5.4). As before, each row corresponds to the convolution of the density functions of a number of independent $\mathcal{N}(10, 2)$ distributions. Similarly, for each of the two numerical convolution routines, there are three sub-columns in Table 6-3(b). The first sub-column is the relative error in expectation of the convolution routine, the second sub-column is the relative error in variance, and the third sub-column is the mean squared error (MSE). After some experimentation, we set $\rho = 1024$.

As is the case in Table 6-1(b) and Table 6-2(b), it is evident that both the direct convolution routine and the FFT-based routine produce accurate results. In both routines, the relative error in expectation and variance is small, although the relative error in expectation is slightly smaller for the direct convolution routine. Note that for direct convolution, the relative error in expectation slightly increases as more random variables are summed. In terms of percentages, no relative error exceeds 0.55% for either convolution routine. The MSE is also small for both numerical convolution routines.

6.1.2 Numerical Minimum

In Chapter 5 we describe the routine we use to calculate the minimum of random variables. If ρ is the number of possible realizations of A or B after A and B have been put on the same scale, then the minimum routine (Algorithm 5.7) runs in $O(\rho^2)$. Like the convolution routines, Algorithm 5.7 takes two random variables as arguments.

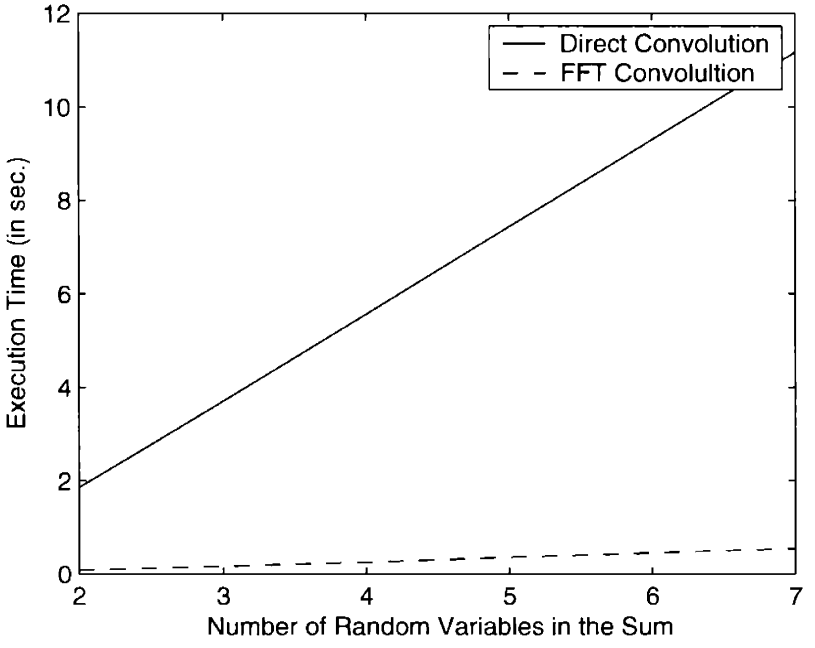


(a) Run-time.

	Type of Convolution						
	Theoretical	Direct			Fast Fourier Transform		
k	$\mathcal{N}(E, Var)$	$\frac{\lvert\hat{E}-E\rvert}{E}$	$\frac{\lvert\widehat{Var}-Var\rvert}{Var}$	MSE	$\frac{\lvert\hat{E}-E\rvert}{E}$	$\frac{\lvert\widehat{Var}-Var\rvert}{Var}$	MSE
2	$\mathcal{N}(20, 4)$	0.000003	0.005387	$< 10^{-8}$	0.000487	0.003440	$< 10^{-7}$
3	$\mathcal{N}(30, 6)$	0.000003	0.005387	$< 10^{-8}$	0.000715	0.002920	$< 10^{-7}$
4	$\mathcal{N}(40, 8)$	0.000003	0.005387	$< 10^{-8}$	0.000602	0.003359	$< 10^{-7}$
5	$\mathcal{N}(50, 10)$	0.000003	0.005387	$< 10^{-9}$	0.000515	0.003680	$< 10^{-7}$
6	$\mathcal{N}(60, 12)$	0.000003	0.005387	$< 10^{-9}$	0.000446	0.003923	$< 10^{-8}$
7	$\mathcal{N}(70, 14)$	0.000003	0.005387	$< 10^{-9}$	0.000391	0.004112	$< 10^{-8}$

(b) Error.

Figure 6-2: Performance of the numerical convolution routines when controlling the interval width. Figure 6-2(a) compares the run-time of the direct convolution routine with the FFT-based convolution routine. Table 6-2(b) compares the relative error in expectation and variance, and the MSE of the two convolution routines.



(a) Run-time.

	Type of Convolution						
	Theoretical	Direct			Fast Fourier Transform		
k	$\mathcal{N}(E, Var)$	$\frac{\lvert\hat{E}-E\rvert}{E}$	$\frac{\lvert\widehat{Var}-Var\rvert}{Var}$	MSE	$\frac{\lvert\hat{E}-E\rvert}{E}$	$\frac{\lvert\widehat{Var}-Var\rvert}{Var}$	MSE
2	$\mathcal{N}(20, 4)$	0.000001	0.005418	$< 10^{-8}$	0.000001	0.005418	$< 10^{-8}$
3	$\mathcal{N}(30, 6)$	0.000001	0.005387	$< 10^{-8}$	0.000651	0.003440	$< 10^{-7}$
4	$\mathcal{N}(40, 8)$	0.000002	0.005333	$< 10^{-8}$	0.000488	0.003860	$< 10^{-7}$
5	$\mathcal{N}(50, 10)$	0.000002	0.005235	$< 10^{-9}$	0.000385	0.004176	$< 10^{-8}$
6	$\mathcal{N}(60, 12)$	0.000007	0.005182	$< 10^{-9}$	0.000316	0.004280	$< 10^{-8}$
7	$\mathcal{N}(70, 14)$	0.000014	0.005060	$< 10^{-9}$	0.000263	0.004271	$< 10^{-8}$

(b) Error.

Figure 6-3: Performance of the numerical convolution routines when controlling the number of possible realizations. Figure 6-3(a) compares the run-time of the direct convolution routine with the FFT-based convolution routine. Table 6-3(b) compares the relative error in expectation and variance, and the MSE of the two convolution routines.

Therefore, to take the minimum of five random variables we need to make four calls to Algorithm 5.7.

To understand the run-time of Algorithm 5.7, we took the minimum of k independent $\mathcal{N}(10, 2)$ random variables, where k varied from 2 to 7. Each distribution had $\min(\mathbf{x}) = 5$ and $\max(\mathbf{x}) = 15$. Furthermore, for each random variable, the number of realizations ρ was set to a uniform random value between 800 and 1800. Figure 6-4(a) displays the run-time of our minimum routine. As is evident, the implementation runs quite well even as the number of random variables increases. Although we could control $\rho^{(C)}$, there is no practical benefit of doing so.

Note that in Figure 6-4(a), the run-time is more linear than quadratic. This is because, as noted in Section 5.2.3, after A and B are put on the same scale, our minimum routine does not further increase $\rho^{(C)}$. Furthermore, in this case, each of the input random variables has a relatively similar number of possible realizations. The practical impact of this property is that Algorithm 5.7 does not slow down much over a series of calls.

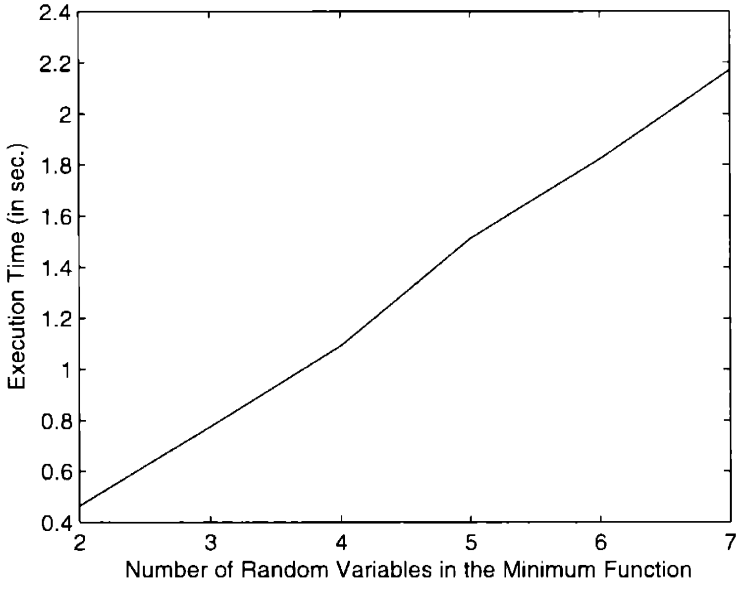
To see the impact of $\rho^{(C)}$ more clearly, we took the minimum of k independent $\mathcal{N}(10, 2)$ random variables a second time. As before, we set $\min(\mathbf{x}) = 5$ and $\max(\mathbf{x}) = 15$. Unlike before, for each random variable, we set the number of realizations ρ to a multiple of 1024 such that the number of possible realizations for each random variable differed much more than before. Figure 6-4(b) displays the run-time of our minimum routine over this new set of random variables. With these random variables Algorithm 5.7 runs closer to its $O(\rho^2)$ complexity.

Note that we do not compare the numerical minimum routine with the theoretical minimum as we are already using the definition of the minimum function to implement Algorithm 5.7.

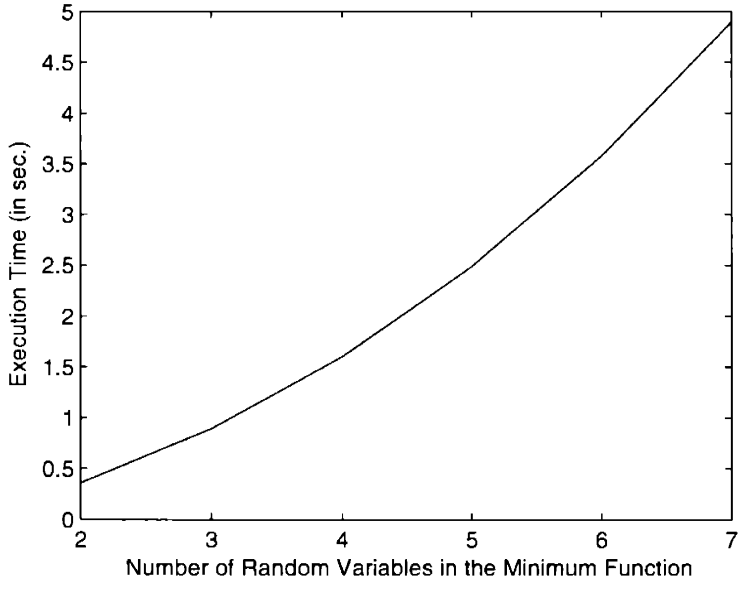
6.1.3 Conclusions

The numerical results for the sum and the minimum routines confirm the following points:

- For a given number of realizations ρ, the FFT-based convolution routine has a faster run-time when compared with the run-time of the direct convolution routine.
- The density functions of the random variables produced by the FFT-based convolution routine are relatively accurate (provided the input random variables are also accurate). Relative to the direct convolution routine, the percent relative error in expectation generally increased with the FFT-based convolution routine, but was still small. The percent relative error in variance generally decreased with the FFT-based convolution routine. The mean squared error increased slightly with the FFT-based convolution routine, but did not exceed 10^{-7}.
- Both methods of controlling the number of possible realizations ρ improve the run-time of both convolution routines.



(a)



(b)

Figure 6-4: Run-time of Algorithm 5.7 over two different sets of random variables. Figure 6-4(a) is the run-time of Algorithm 5.7 where, for each random variable, the number of realizations ρ was set to a uniform random value between 800 and 1800. Figure 6-4(b) is the run-time of Algorithm 5.7 where, for each random variable, the number of realizations ρ was set to a multiple of 1024.

- As implemented in Algorithm 5.7, the minimum routine runs well. There does not appear to be any need to control the number of possible realizations ρ to improve run-time.

In small networks it is not necessary to have efficient sum and minimum routines to implement our routing procedure. However, in the larger networks that are typical of practical applications, the efficiency of the sum and the minimum routine is the critical component in determining the run-time of the minimum travel time distribution algorithms described in Chapter 4.

6.2 Routing on Example Networks

Our basic routing approach is described in Algorithm 2.3. Chapter 4 discusses **Step 2** and Chapter 5 discusses **Step 1** of Algorithm 2.3. Chapter 5 presents several methods to generate a $\mathcal{TT}$-$\mathcal{DAG}$ G' from a possibly cyclic graph G. For the example networks in this section, we use Algorithm 5.3 to generate G'. Chapter 5 also presents theoretically efficient implementations of the sum and minimum functions of random variables. Section 6.1 demonstrates the practical efficiency of these implementations. Of the two convolution routines discussed in Chapter 5, we use the Fast Fourier Transform convolution routine for the example networks in this section. Furthermore, the number of possible realizations ρ for each random variable is not controlled.

Now that we have all the components of Algorithm 2.3, we illustrate how this algorithm works on three different networks. The first network is a small, simple network. The second network is a small, but more complicated network. Relative to these two small networks, the third network is much larger. It is composed of 200 nodes and 522 arcs. This third network is a modified model of the Amsterdam A-10 beltway.

The results in this section were obtained by simulating each of the three networks using a network simulator. All algorithms were implemented in C++. The machine used to obtain the results in this section was a 933 MHz Pentium 3 CPU with 256M of RAM, and the Linux 2.4 kernel.

6.2.1 A Small Network

Consider the small network depicted in Figure 6-5(a). All arcs have travel times distributed according to $\mathcal{N}(10, 2)$ or $\mathcal{N}(10, 8)$, except for arcs $(2, 4), (4, 2)$ and arcs $(10, 11), (11, 10)$, all of which have a constant travel time of 0.1. Strictly speaking, the arcs $(2, 4), (4, 2), (11, 10)$, and $(10, 11)$ are not necessary; these arcs are just used to delineate node 2 and node 11 from the rest of the network since node 2 and node 11 are the source and sink nodes in our simulation. Similarly, the exact travel time along arcs $(2, 4), (4, 2), (11, 10)$, and $(10, 11)$ has no effect in our routing procedure since a user destined for node 2 must always traverse arc $(4, 2)$, and a user destined for node 11 must always traverse arc $(10, 11)$. For illustrative purposes, we set the travel time along these arcs to be 0.1.

The first thing to notice about this network is its structure. Formally it is not a series-parallel network. However, informally, we see the underlying structure has several series and parallel components. Therefore, the $\mathcal{TT}$-$\mathcal{DAG}$s we generate for routing will, for the most part, be series-parallel. As a consequence, it is relatively easy to calculate the minimum travel time distributions on this network.

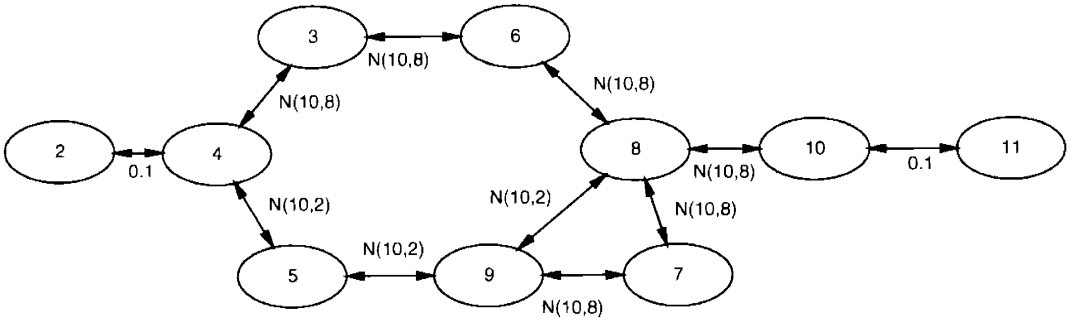
Suppose a user located at node 4 is destined for node 11. What node $j \in N^+(4)$ should the user be routed on? Applying Algorithm 2.3, we see that since $N^+(4) = \{3, 5\}$, we need to calculate $P^*_{3,11}$ and $P^*_{5,11}$.

First, we consider the calculation of $P^*_{5,11}$. Before we can calculate $P^*_{5,11}$, we need to generate a $\mathcal{TT}$-$\mathcal{DAG}$ subgraph G' of G from node 5 to node 11. Using Algorithm 5.3, the resulting $\mathcal{TT}$-$\mathcal{DAG}$ G' from node 5 to node 11 is presented in Figure 6-5(b). Note that G' is series-parallel.

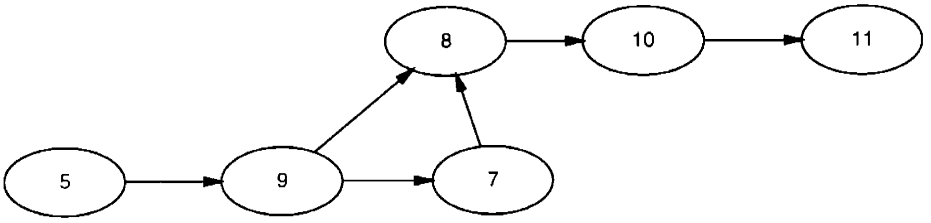
Now that we have a $\mathcal{TT}$-$\mathcal{DAG}$ G', we need to check whether G' is series-parallel. This is done with Algorithm 4.2. In this case, it is a trivial observation to see that G' is series-parallel. Now that G' has been identified as a series-parallel graph, we can calculate $P^*_{5,11}$ by running Algorithm 4.3 on G'. We initialize the algorithm by adding all nodes to Q except for node 5 and node 11 so that $Q = \{9, 7, 8, 10\}$. Node 9 is selected as the first node to perform series and parallel reductions on. In **Step 1** of Algorithm 4.3, we attempt to remove all outgoing parallel arcs from node 9, but there are none. In **Step 2** of Algorithm 4.3, we attempt to remove all incoming parallel arcs to node 9, but there are also none. In **Step 3** of Algorithm 4.3, we check to see if $|N^+(9)| = 1$ and if $|N^-(i)| = 1$. Since only $|N^-(i)| = 1$, we do nothing and instead select the next node (node 7) from Q. Like node 9, node 7 does not have any incoming or outgoing parallel arcs. However, in **Step 3** of Algorithm 4.3, since node 7 has one incoming arc and one outgoing arc, we can perform a series reduction. Arcs $(9, 7)$ and $(7, 8)$ are replaced by a new arc $(9, 8)$ with travel time equal to $X_{98} = X_{97} + X_{78}$. Since X_{97} is distributed as $\mathcal{N}(10, 8)$ and X_{78} is distributed as $\mathcal{N}(10, 8)$, then X_{97} is distributed as the sum of $\mathcal{N}(10, 8)$ and $\mathcal{N}(10, 8)$, which equals $\mathcal{N}(20, 16)$. Node 9 is also added back to Q.

Now there are two parallel arcs from node 9 to node 8 and $Q = \{8, 10, 9\}$. Node 8 is selected from Q. Node 8 has no outgoing parallel arcs, but has two incoming parallel arcs from node 9; one from the original network, and one added from the prior series reduction of node 7. These two arcs from node 9 to node 8 are replaced with a single new arc $(9, 8)$ that has travel time $\mathcal{MIN}(X^1_{89}, X^2_{89}) = \mathcal{MIN}(\mathcal{N}(20, 16), \mathcal{N}(10, 2))$. Note that nearly all possible realizations of $\mathcal{N}(10, 2)$ are better than the smallest realizations of $\mathcal{N}(20, 16)$, which means that the difference between $\mathcal{MIN}(\mathcal{N}(20, 16), \mathcal{N}(10, 2))$ and $\mathcal{N}(10, 2)$ is small. For the purposes of this example, we take $\mathcal{MIN}(\mathcal{N}(20, 16), \mathcal{N}(10, 2)) = \mathcal{N}(10, 2)$.

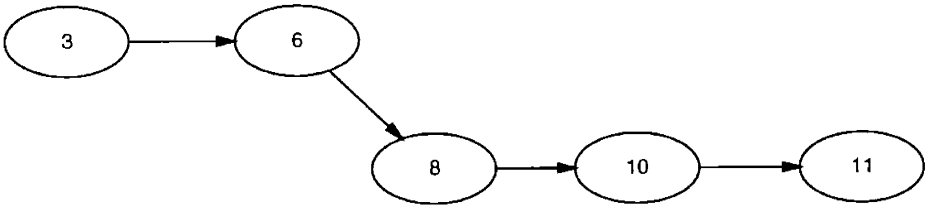
In **Step 3** of Algorithm 4.3, we see that node 8 now has one incoming arc and one outgoing arc, so a series reduction can be performed. Arcs $(9, 8)$ and $(8, 10)$ are replaced by a new arc $(9, 10)$ with travel time $\mathcal{N}(20, 10)$ which is the sum of $\mathcal{N}(10, 2)$ and $\mathcal{N}(10, 8)$. Q is now $\{10, 9\}$. Node 10 is selected and has both a single incoming arc and a single outgoing arc. So in **Step 3** of Algorithm 4.3, arcs $(9, 10)$ and $(10, 11)$ are replaced with a single new arc $(9, 11)$ with travel time $\mathcal{N}(20.1, 10)$ since arc $(10, 11)$ has travel time 0.1. Node 11 is not added to Q since node $11 = t$. Q is now $\{9\}$.



(a) The Original Network.



(b) Subgraph Generated from Node 5 to Node 11.



(c) Subgraph Generated from Node 3 to Node 11.

Figure 6-5: The Subgraphs Generated by Algorithm 5.3 on Figure 6-5(a) for a User Located at Node 4 destined for Node 11.

Iteration	Q	Node Selected (i)	$N^+(i)$	$N^-(i)$
0	$\{9, 7, 8, 10\}$	9	2	1
1	$\{7, 8, 10\}$	7	1	1
2	$\{8, 10, 9\}$	8	1	2
3	$\{10, 9\}$	10	1	1
4	$\{9\}$	9	1	1

Table 6.1: The Calculation of $P^*_{5,11}$ with Algorithm 4.3.

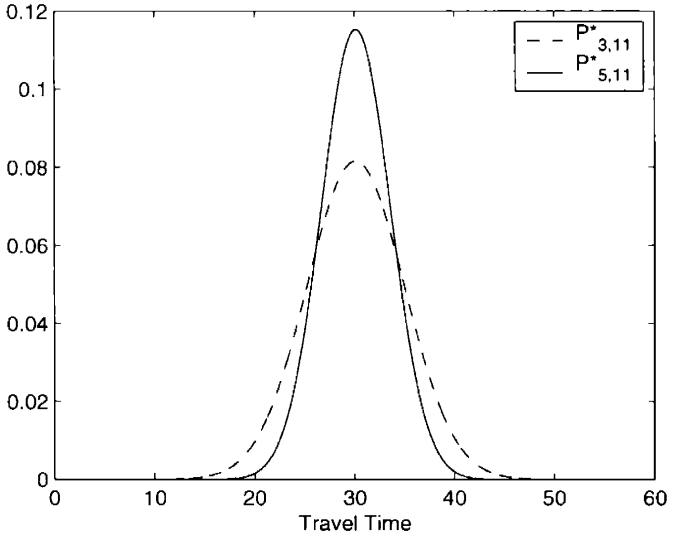


Figure 6-6: Comparison of the Density Functions of $P^*_{3,11}$ and $P^*_{5,11}$.

Node 9 is selected and has a single incoming and single outgoing arc. Arcs $(5, 9)$ and $(9, 11)$ are replaced with a new arc $(5, 11)$ with travel time $\mathcal{N}(30.1, 12)$, which is the sum of $\mathcal{N}(10, 2)$ and $\mathcal{N}(20.1, 10)$. This is $P^*_{5,11}$. A summary of the calculation of $P^*_{5,11}$ is listed in Table 6.1.

To use Algorithm 2.3, we also need $P^*_{3,11}$. The subgraph from node 3 to node 11 is the simple path of nodes in series in Figure 6-5(c). We note $P^*_{3,11}$ can simply be evaluated as $P^*_{3,11} = X_{36} + X_{68} + X_{8,10} + X_{10,11}$, which yields the distribution $\mathcal{N}(30.1, 24)$. $P^*_{3,11}$ and $P^*_{5,11}$ are plotted in Figure 6-6. We see that both $P^*_{3,11}$ and $P^*_{5,11}$ have the same expectation, but $P^*_{5,11}$ has less variance. Also note that $E[P^*_{3,11}] = 30.1$ and $E[P^*_{5,11}] = 30.1$, which is consistent with the result in Theorem 4.

Now that $P^*_{5,11}$ and $P^*_{3,11}$ are known, the user can select a routing objective. For example, suppose the user is interested in taking a path from node 4 to node 11 that has a low expected travel time with low variance. If we take $\Phi(X_{ij}) = E[X_{ij}] + \theta \cdot Var(X_{ij})$ and $\Psi(P^*_{jt}) = E[P^*_{jt}] + \theta \cdot Var(P^*_{jt})$, then we have the following equation to describe the routing objective:

$$j^* \in \underset{j \in N^+(i)}{\operatorname{argmin}} (E[X_{ij}] + \theta \cdot Var(X_{ij}) + E[P^*_{jt}] + \theta \cdot Var(P^*_{jt})) \tag{6.1}$$

where θ is a user-defined parameter such that $\theta \in \mathbb{R}^+ \cup \{0\}$. From an optimization perspective, θ can be interpreted as a penalty or weight that reflects the importance of variance in travel time to the user. From a behavioral perspective, θ reflects the user-perceived disutility of variance in travel time. In other words, if a user values low variance in travel time, the user should choose a large value of θ to reflect the relative disutility of high variance in travel time. If the user is indifferent to variance in travel time, then θ should be set to 0.

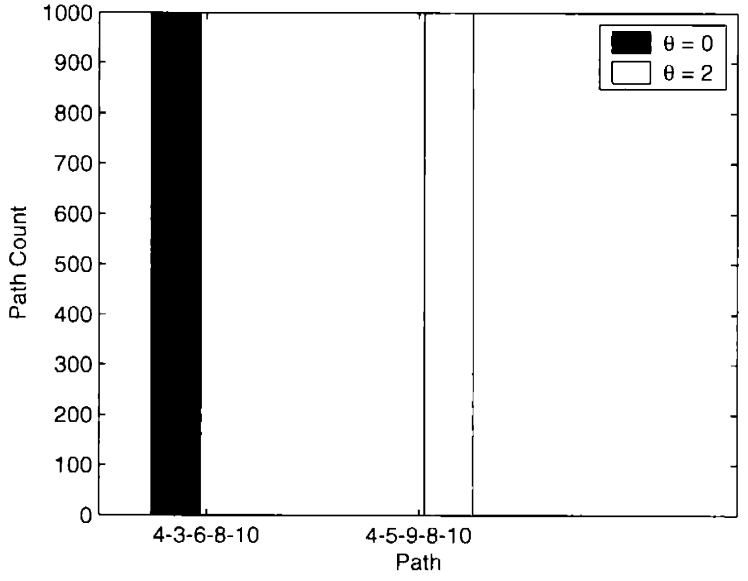
In the case where $\theta = 0$, the routing decisions are based solely on expectation. A higher value of θ will generally use paths with lower variance. For adjacent node 3 and $\theta = 0$, we have $E[X_{43}] + \theta \cdot Var(X_{43}) + E[P^*_{3,11}] + \theta \cdot Var(X_{3,11}) = 10 + 30.1 = 40.1$. For adjacent node 5 and $\theta = 0$, we have $E[X_{45}] + \theta \cdot Var(X_{45}) + E[P^*_{5,11}] + \theta \cdot Var(X_{5,11}) = 10 + 30.1 = 40.1$. Although in our simulation node 3 is selected, it also could have been node 5. In the case where $\theta = 2$, node 3 yields $10 + 2(8) + 30.1 + 2(24) = 104.1$ and node 5 yields $10 + 2(2) + 30.1 + 2(12) = 68.1$. Thus, node 5 will be selected due to its lower variance (and since nodes 3 and 5 have the same expectation). Also note that with this routing objective, the same node j will always be selected for a particular source i and sink t. In this sense, this calculation can be performed off-line.

In the simulation, there was a source-sink pair from node 2 to node 11 and a source-sink pair from node 11 to node 2. The simulation ran ten times for each value of θ; each run contained a different seed for the random number generator. In each run, 100 users traveled between each source-sink pair (200 users total in the network). Therefore, over ten runs, each source-sink pair had a total of 1000 users. Figure 6-7(a) shows the paths traversed from node 2 to node 11 under the two different values of θ.

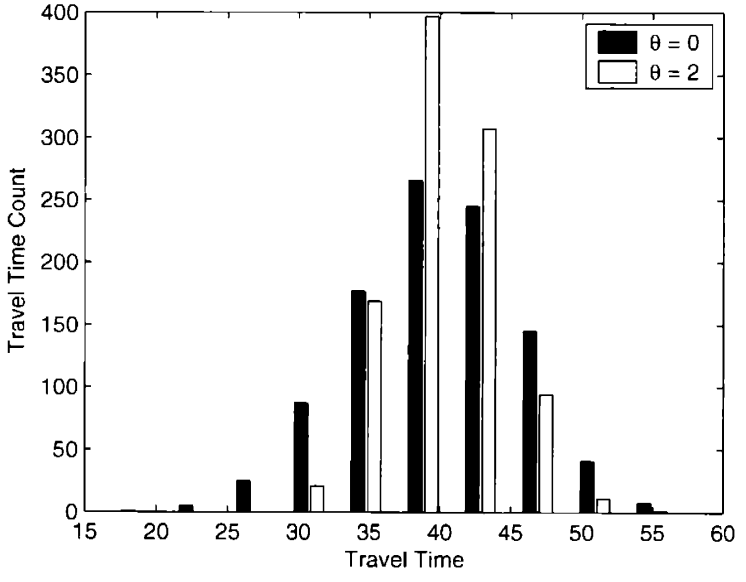
Next, consider the actual realized travel times from node 2 to node 11 in Figure 6-7(b). When $\theta = 0$, the sample expectation $\hat{E}$ of the actual travel time from node 2 to node 11 is 39.8117. The corresponding sample variance $\widehat{Var}$ is 33.4444. When $\theta = 2$, the sample expectation $\hat{E}$ of the actual travel time from node 2 to node 11 is 40.0890. The corresponding sample variance $\widehat{Var}$ is 14.3420. Relative to the expectation and variance when $\theta = 0$, this represents a 0.7% increase in expectation and a 57.1% decrease in the variance of the actual travel time.

As expected, the distribution of travel times for $\theta = 2$ has less variance than when $\theta = 0$. This is important because a decision based solely on expectation does not provide the user with enough information to make a more informed choice when travel times are random. For example, if a user is at node 4 and it is very important for the user to arrive at node 11 in approximately 40 time units, the user would want to select adjacent node 5 (corresponding to $\theta = 2$) since there is less variance in travel time. On the other hand, if a user is interested in a travel time of less than 30 *and* is willing to accept the possibility of a travel time greater than 50, then the user would want to select node 3, which corresponds to $\theta = 0$. Note that there is almost no possibility of achieving a travel time of less than 30 by selecting node 5. Intuitively, this corresponds to the classic concept of "risk versus reward"; the user must accept the risk of a very high travel time if the user wants the reward of a very low travel time.

To show the flexibility of this approach to routing, suppose we have a system



(a) Paths Traversed. Each path begins at node 2 and ends at node 11.



(b) Travel Times.

Figure 6-7: Routing performance from node 2 to node 11 on Figure 6-5(a) using the routing objective in Equation 6.1 with two different θs. Figure 6-7(a) displays the paths traversed and Figure 6-7(b) displays the corresponding travel times. For $\theta = 0, \hat{E}[D_{2,11}] = 39.8117$ and $\widehat{Var}(D_{2,11}) = 33.4444$. For $\theta = 2, \hat{E}[D_{2,11}] = 40.0890$ and $\widehat{Var}(D_{2,11}) = 14.3420$.

where the travel times on the adjacent arcs are known immediately before the arc is traversed (for example, see [15]). In a transportation network, such information might come from an Intelligent Transportation System (ITS). In this case, we can take $\Phi(X_{ij}) = x_{ij}$ (the actual travel time realization) and $\Psi(P^*_{jt}) = E[P^*_{jt}] + \theta \cdot Var(P^*_{jt})$. The routing objective is similar to the objective in Equation 6.1 except for the use of the realizations of the travel time of the adjacent arcs:

$$j^* \in \underset{j \in N^+(i)}{\operatorname{argmin}}(x_{ij} + E[P^*_{jt}] + \theta \cdot Var(P^*_{jt})) \tag{6.2}$$

For each $j \in N^+(i)$, we can interpret this equation as a routing objective that is conditioned on the known travel time x_{ij}. We can interpret this conditioning as a shift of P^*_{jt} by x_{ij}. This means that $E[P^*_{jt}+x_{ij}] = E[P^*_{jt}]+x_{ij}$ and $Var(P^*_{jt}+x_{ij}) = Var(P^*_{jt})$, which agrees with Equation 6.2. In this sense, the knowledge of x_{ij} only affects the expected travel time to node t through node j: it does not affect the variance of the travel time to node t through node j.

When $\theta = 0$ in the routing objective described by Equation 6.1, we only select $j^* = 3$. With the routing objective described by Equation 6.2, both node 3 and node 5 can be selected depending on the realization of x_{ij}. If, from the realizations, we have $x_{43} > x_{45}$, then we know $x_{43} + E[P^*_{3,11}] > x_{45} + E[P^*_{5,11}]$. A similar equation follows for $x_{43} < x_{45}$. For $\theta = 2$, node 5 is still always selected. In other words, no realization of x_{43} can convince the user to select node 3.

Figure 6-8(a) shows the paths traversed when routing with Equation 6.2. The associated actual travel times are also presented in Figure 6-8(b). When $\theta = 0$, the sample expectation $\hat{E}$ of the actual travel time from node 2 to node 11 is 38.8874. The corresponding sample variance $\widehat{Var}$ is 22.2661. When $\theta = 2$, the sample expectation $\hat{E}$ of the actual travel time from node 2 to node 11 is 40.0348. The corresponding sample variance $\widehat{Var}$ is 13.4408. Relative to the expectation and variance when $\theta = 0$, this represents a 3.0% increase in expectation and a 39.6% decrease in the variance of the actual travel time.

With Equation 6.2, the actual travel times produced when $\theta = 0$ are more similar to the travel times to $\theta = 2$. This follows from the fact that when $\theta = 0$, node 5 is selected approximately half the time. Note that a realization of x_{97} was so small, it resulted in a user taking a path with a high expected travel time (path 2-4-5-9-7-8-10-11).

Note that when routing with Equation 6.2 and $\theta = 0$ the expected travel time decreased 2.3% relative to the expected travel time when routing with Equation 6.1 and $\theta = 0$. There are almost no realizations larger than 45. Similarly, the variance of the actual travel time decreased by 33.4%. These percentages represent a certain *value of information*; in this case, the knowledge of the adjacent arc realizations appears to allow the user to decrease the variance in travel time *even* when the user is indifferent to the variance in travel time ($\theta = 0$). When $\theta = 2$, the same path was always traversed for both Equation 6.1 and Equation 6.2 and there is minimal change in the expected travel time (0.1% decrease) or variance of travel time (6.3% decrease). In this case, the relative disutility of high variance in travel time cannot

be offset by the knowledge of the adjacent arc realizations.

Although the subgraphs in Figure 6-5(b) and Figure 6-5(c) are series-parallel graphs, it is not necessarily the case that Algorithm 5.3 will *always* generate a series-parallel graph for this network. If we consider generating a $\mathcal{TT}$-$\mathcal{DAG}$ from node 7 to node 2, Algorithm 5.3 generates the non-series-parallel subgraph in Figure 6-9.

If we calculate P^*_{72} on Figure 6-9 using Algorithm 4.4, we get the dashed curve in Figure 6-10. (An illustration of the execution of Algorithm 4.4 is presented in Section 6.2.2). The other three curves represent the possible path travel time distributions from node 7 to node 2. For example, path 7-9-5-4-2 has a travel time of $\mathcal{N}(30.1, 12)$, path 7-8-9-5-4-2 has a travel time of $\mathcal{N}(40.1, 14)$, and path 7-8-6-3-4-2 has a travel time of $\mathcal{N}(40.1, 32)$.

Note that Figure 6-10 is consistent with Theorem 5. It also provides intuition as to how P^*_{72} encapsulates the travel time from node 7 to node 2.

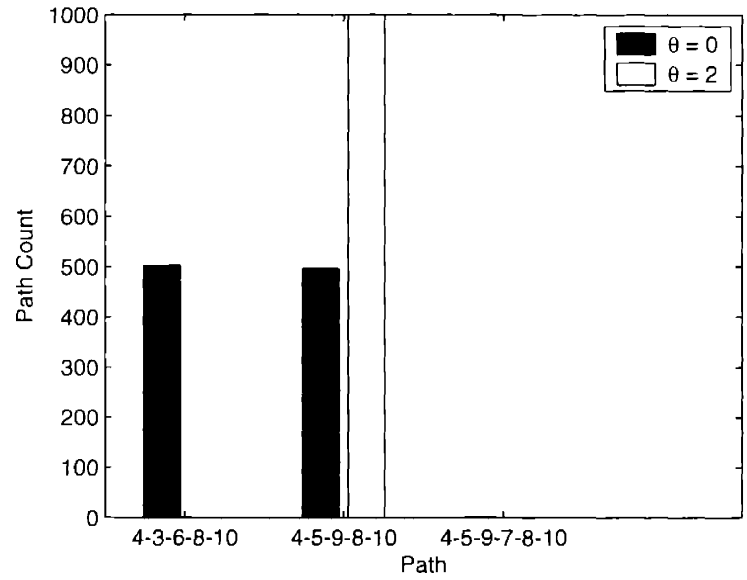
6.2.2 A Small, More Complicated Network

The simple structure of the network in Figure 6-5(a) makes it useful for illustrative purposes. But what is the routing performance in a slightly more complicated network? Consider the network shown in Figure 5-1. Suppose a user is located at node 3 and is destined for node 10. Since $N^+(3) = \{4, 5, 6\}$, three acyclic subgraphs need to be generated. These subgraphs are depicted in Figure 6-11(a), Figure 6-11(b), and Figure 6-11(c).

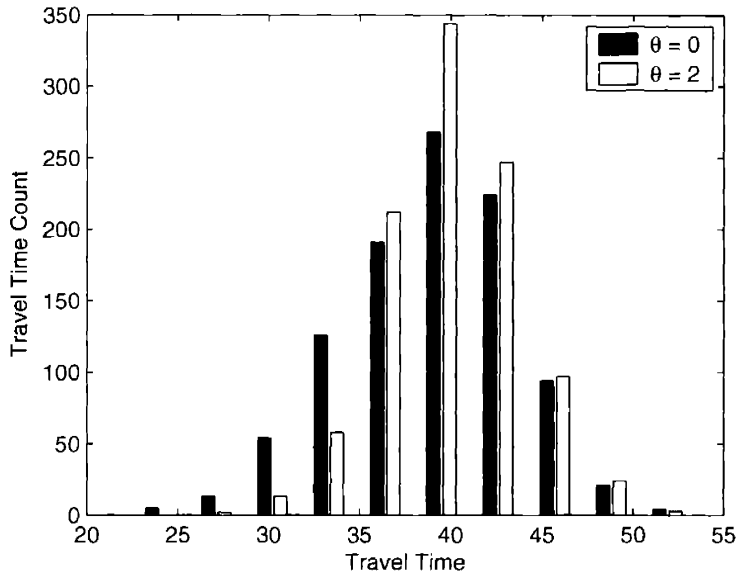
Since $N^+(3) = \{4, 5, 6\}$, we need to calculate $P^*_{4,10}$, $P^*_{5,10}$, and $P^*_{6,10}$. We consider $P^*_{5,10}$ first. The graph corresponding to this calculation is series-parallel and is depicted in Figure 6-11(b).

For $P^*_{5,10}$, we initialize the algorithm by adding all nodes to Q except for node 5 and node 10 so that $Q = \{9, 7, 8\}$. Node 9 is selected as the first node to reduce. For node 9, we see that $|N^-(9)| = 1$ and $|N^+(9)| = 1$, so node 9 is series reducible. We replace arcs $(5, 9), (9, 8)$ with a new arc $(5, 8)^2$ (now there are two arcs from node 5 to node 8). We also set the travel time $X^2_{58} = \mathcal{N}(10, 8) + \mathcal{N}(10, 8) = \mathcal{N}(20, 16)$. The next node in Q is node 7. Like node 9, we have $|N^-(7)| = 1$ and $|N^+(7)| = 1$, so node 7 is also series reducible. We replace arcs $(5, 7), (7, 8)$ with a new arc $(5, 8)^3$ (now there are three arcs from node 5 to node 8). We also set the travel time $X^3_{58} = \mathcal{N}(10, 8) + \mathcal{N}(10, 2) = \mathcal{N}(20, 10)$. Now, node 8 is selected since it is the only node remaining in Q. There are three incoming arcs to node 8 from node 5, so we perform a parallel reduction and replace $(5, 8)^1, (5, 8)^2$, and $(5, 8)^3$ with a single arc $(5, 8)$. The travel time of X_{58} is set to $\mathcal{MIN}(X^1_{58}, X^2_{58}, X^3_{58})$. After this parallel reduction is performed, node 8 has $|N^-(8)| = 1$ and $|N^+(8)| = 1$. We can then perform a series reduction with arcs $(5, 8)$ and $(8, 10)$ so the network only contains a single arc $(5, 10)$. $P^*_{5,10} = X_{5,10} = X_{58} + X_{8,10} = \mathcal{MIN}(X^1_{58}, X^2_{58}, X^3_{58}) + 0.1$. Note that although the ordering of the elements in Q determines a particular order of operations, we will find the same $P^*_{5,10}$ for any ordering.

Unlike the previous section, we cannot solve $\mathcal{MIN}(X^1_{58}, X^2_{58}, X^3_{58})$ by inspection. Figure 6-12 compares $P^*_{5,10}$ with the travel times of several paths from node 5 to node 10. With respect to Figure 6-11(b), there are three possible paths from node 5 to node



(a) Paths Traversed. Each path begins at node 2 and ends at node 11.



(b) Travel Times.

Figure 6-8: Routing performance from node 2 to node 11 on Figure 6-5(a) using the routing objective in Equation 6.2 with two different θs. Figure 6-8(a) displays the paths traversed and Figure 6-8(b) displays the corresponding travel times. For $\theta = 0$, $\hat{E}[D_{2,11}] = 38.8874$ and $\widehat{Var}(D_{2,11}) = 22.2661$. For $\theta = 2$, $\hat{E}[D_{2,11}] = 40.0348$ and $\widehat{Var}(D_{2,11}) = 13.4408$.

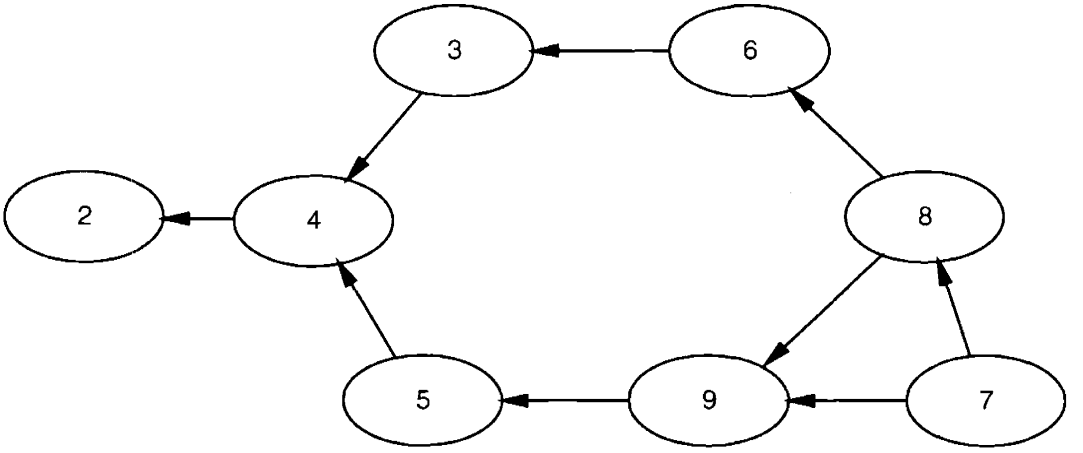


Figure 6-9: The Subgraph Generated by Algorithm 5.3 on Figure 6-5(a) from Node 7 to Node 2.

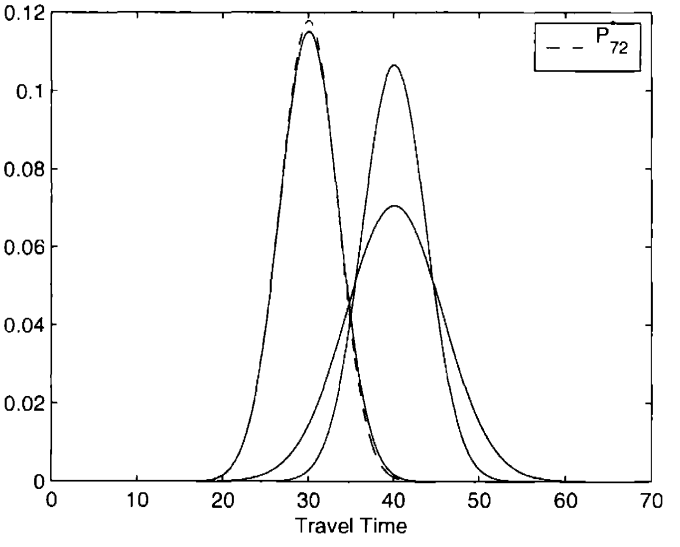
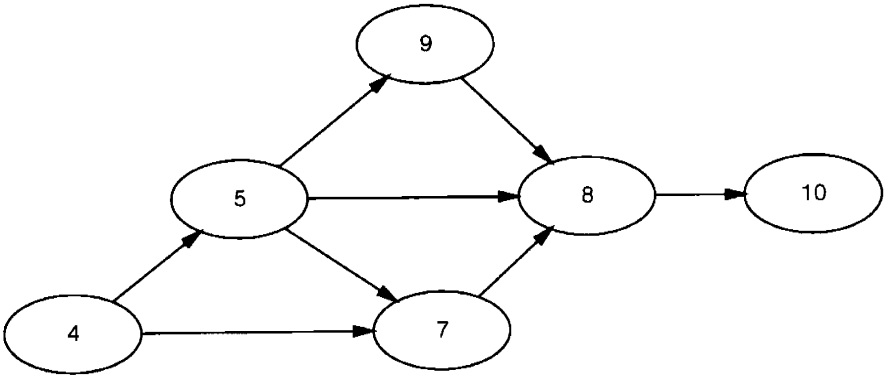
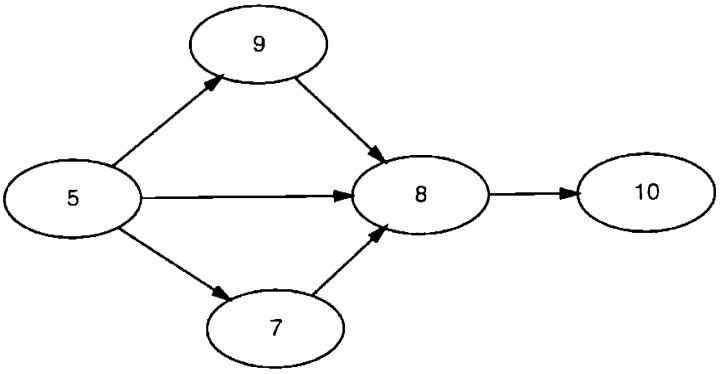


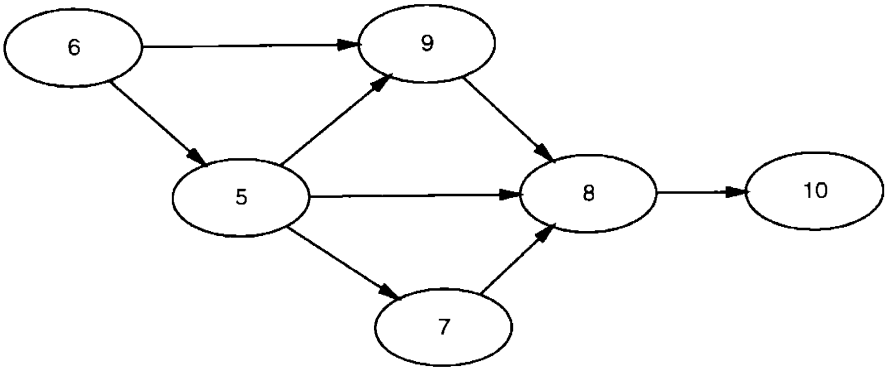
Figure 6-10: Comparison of travel time density functions from node 7 to node 2. The dashed curve is the density function of the minimum travel time distribution P^*_{72}. The other three curves are the travel time density functions of the possible paths from node 7 to node 2 on Figure 6-9. For example, the travel time along path 7-9-5-4-2 is distributed as $\mathcal{N}(30.1, 12)$, the travel time along path 7-8-9-5-4-2 is distributed as $\mathcal{N}(40.1, 14)$, and the travel time along path 7-8-6-3-4-2 is distributed as $\mathcal{N}(40.1, 32)$.



(a) Subgraph Generated from Node 4 to Node 10.



(b) Subgraph Generated from Node 5 to Node 10.



(c) Subgraph Generated from Node 6 to Node 10.

Figure 6-11: The Subgraphs Generated by Algorithm 5.3 on Figure 5-1 for a User Located at Node 3 destined for Node 10.

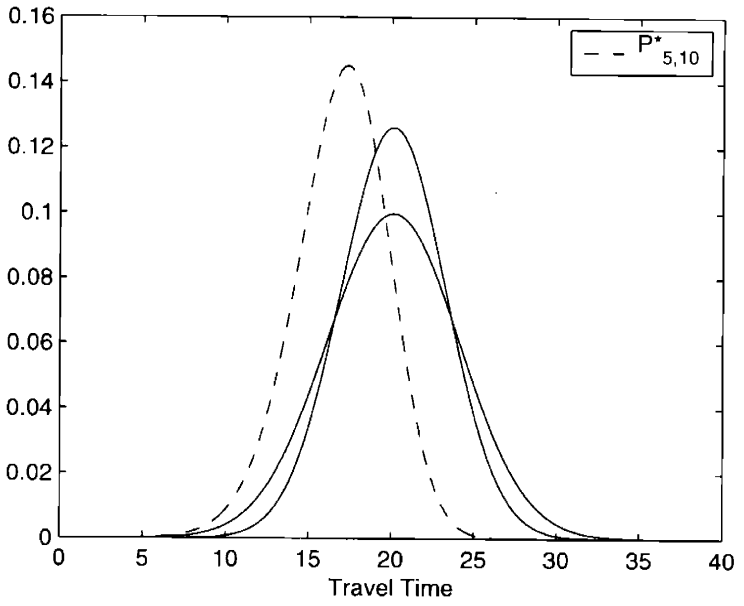


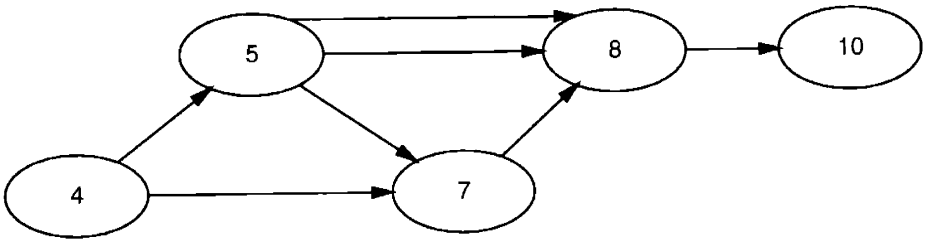
Figure 6-12: Comparison of travel time density functions from node 5 to node 10. The dashed curve is density function of the minimum travel time distribution $P^*_{5,10}$. Of the two remaining curves, one curve is the density function of the travel time along path 5-7-8-10 ($\mathcal{N}(20.1, 10)$). The other curve is the density function of the travel time along both path 5-9-8-10 and path 5-8-10 ($\mathcal{N}(20.1, 16)$).

10; 5-9-8-10, 5-8-10, and 5-7-8-10. Both path 5-9-8-10 and path 5-8-10 have a travel time distribution of $\mathcal{N}(20.1, 16)$, while path 5-7-8-10 has a travel time distribution of $\mathcal{N}(20.1, 10)$. We note that $E[P^*_{5,10}]$ is less than the expected value of any other path from node 5 to node 10 in Figure 6-11(b). This is consistent with Theorem 4.

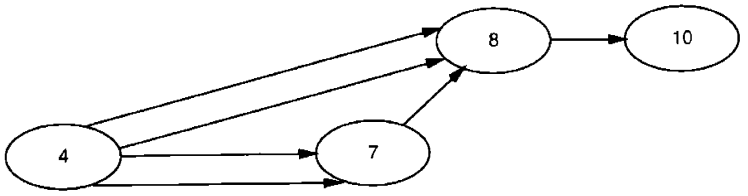
Next, we need to calculate $P^*_{4,10}$. We first see the subgraph in Figure 6-11(a) is not series-parallel reducible. Because of this, we need to use Algorithm 4.4 to approximate $P^*_{4,10}$. **Step 1** of Algorithm 4.4 performs a series reduction with arc $(5, 9)$ and arc $(9, 8)$. A new arc $(5, 8)^2$ is added with $X^2_{58} = \mathcal{N}(20, 16)$. The resulting graph is in Figure 6-13(a). Although there are two parallel arcs from node 5 to node 8, the algorithm does not perform a parallel reduction since $N^+(5) \geq 2$. This is the short-circuit discussed in Section 4.2.1.

Since $|N'| > 2$, we need to find and condition on a Type-1 or Type-2 arc. Arc $(4, 5)$ is a Type-1 arc, so we set X_{45} to $E[X_{45}]$. Now we can remove arc $(4, 5)$ and add new arcs $(4, 8)^1, (4, 8)^2$, and $(4, 7)^2$. X^1_{48} is set to $E[X_{45}] + X^1_{58} = \mathcal{N}(30, 16)$. $X^2_{48} = E[X_{45}] + X^2_{58} = \mathcal{N}(30, 16)$, and X^2_{47} is set to $E[X_{45}] + X_{57} = \mathcal{N}(20, 8)$. The resulting network is shown in Figure 6-13(b).

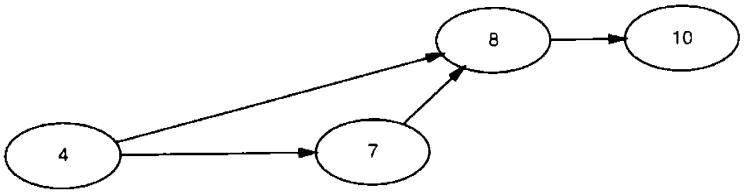
At this point, the graph has now become series-parallel reducible. In the next execution of **Step 1** of Algorithm 4.4, G' will be completely reduced to a single arc $(4, 10)$. Arcs $(4, 8)^1, (4, 8)^2$ are removed with a parallel reduction and replaced with a new arc $(4, 8)$ with $X_{48} = \mathcal{MIN}(X^1_{48}, X^2_{48}) = \mathcal{MIN}(\mathcal{N}(30, 16), \mathcal{N}(30, 16))$. Arcs $(4, 7)^1, (4, 7)^2$ are also removed with a parallel reduction and replaced with a new



(a) Figure 6-11(a) after the Removal of Node 9.



(b) Figure 6-13(a) after the Removal of Node 5.



(c) Figure 6-13(b) after the Removal the Parallel Arcs.

Figure 6-13: The Application of Algorithm 4.4 to Figure 6-11(a).

arc $(4,7)$ with $X_{47} = \mathcal{MIN}(X_{47}^1, X_{47}^2) = \mathcal{MIN}(\mathcal{N}(10,2), \mathcal{N}(20,8)) \approx \mathcal{N}(10,2)$. Figure 6-13(c) shows the network after these parallel reductions.

Arcs $(4,7)$ and $(7,8)$ are in series so we perform a series reduction and add the new arc $(4,8)^2$ with $X_{48}^2 = X_{47} + X_{78} = \mathcal{MIN}(\mathcal{N}(10,2), \mathcal{N}(20,8)) + \mathcal{N}(10,8) \approx \mathcal{N}(10,2) + \mathcal{N}(10,8) = \mathcal{N}(20,10)$. Now arcs $(4,8)^1$ and $(4,8)^2$ are in parallel so they are removed and arc $(4,8)$ is added with $X_{48} = \mathcal{MIN}(\mathcal{N}(20,10), \mathcal{MIN}(\mathcal{N}(30,16), \mathcal{N}(30,16)))$. Finally, arc $(4,8)$ is in series with arc $(8,10)$ so arcs $(4,8), (8,10)$ are replaced with a new arc $(4,10)$ with $X_{4,10} = X_{48} + 0.1$. Since arc $(4,10)$ is the only arc left $X_{4,10}$ is $P_{4,10}^*$.

The density function of the minimum travel time distribution $P_{4,10}^*$ is shown with a dashed line in Figure 6-14. The other curves in Figure 6-14 are the density functions of travel time for the possible paths from node 4 to node 10 in Figure 6-11(a). These paths are 4-5-9-8-10, 4-5-8-10, 4-5-7-8-10, and 4-7-8-10.

By symmetry, the calculation of $P_{6,10}^*$ involves the same order of operations as for $P_{4,10}^*$. However, because the distributions are not the same, $P_{6,10}^*$ is different from $P_{4,10}^*$. Figure 6-15 shows the minimum travel time distribution $P_{6,10}^*$ as a dashed line.

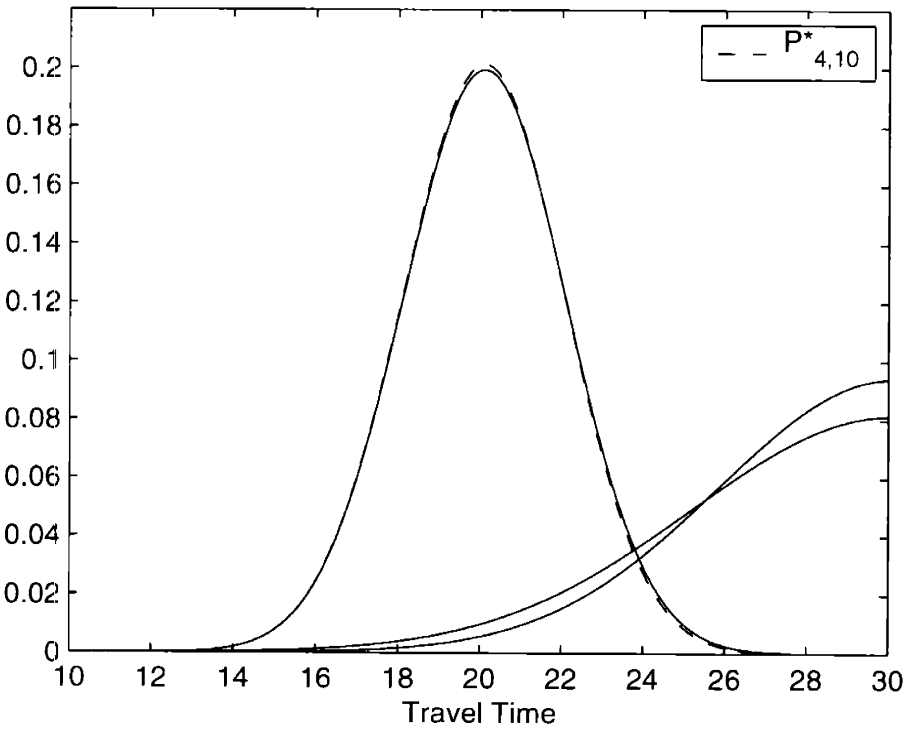


Figure 6-14: Comparison of travel time density functions from node 4 to node 10. The dashed curve is the density function of the minimum travel time distribution $P^*_{4,10}$. The three remaining curves correspond to the travel time density functions of path 4-5-9-8-10 and path 4-5-8-10 ($\mathcal{N}(30.1, 24)$), path 4-5-7-8-10 ($\mathcal{N}(30.1, 18)$), and path 4-7-8-10 ($\mathcal{N}(20.1, 4)$).

Like Figure 6-14, the other curves represent the density functions of travel time of the other possible paths from node 6 to node 10 in Figure 6-11(c). These paths are 6-5-7-8-10, 6-5-8-10, 6-5-9-8-10, and 6-9-8-10.

Now that $P^*_{4,10}$, $P^*_{6,10}$, and $P^*_{5,10}$ are known, the user selects a routing objective. As before, we consider the two different routing objectives corresponding to Equation 6.1 and Equation 6.2.

In the simulation, there was a source-sink pair from node 2 to node 10 and a source-sink pair from node 10 to node 2. The simulation ran ten times for each value of θ; each run contained a different seed for the random number generator. In each run, 100 users traveled between each source-sink pair (200 users total in the network). Therefore, over ten runs, each source-sink pair had a total of 1000 users.

Figure 6-16(a) shows the paths traversed when routing according to Equation 6.1 and two different values of θ. The actual realized travel times are presented in Figure 6-16(b). When $\theta = 0$, the sample mean of the actual travel time $\hat{E}[D_{2,10}]$ is 30.0179. The corresponding sample variance $\widehat{Var}(D_{2,10})$ is 23.1403. For $\theta = 2$, the sample mean of the actual travel time $\hat{E}[D_{2,10}]$ is 29.9877. The corresponding sample variance $\widehat{Var}(D_{2,10})$ is 5.5293. Relative to the expectation and variance when $\theta = 0$,

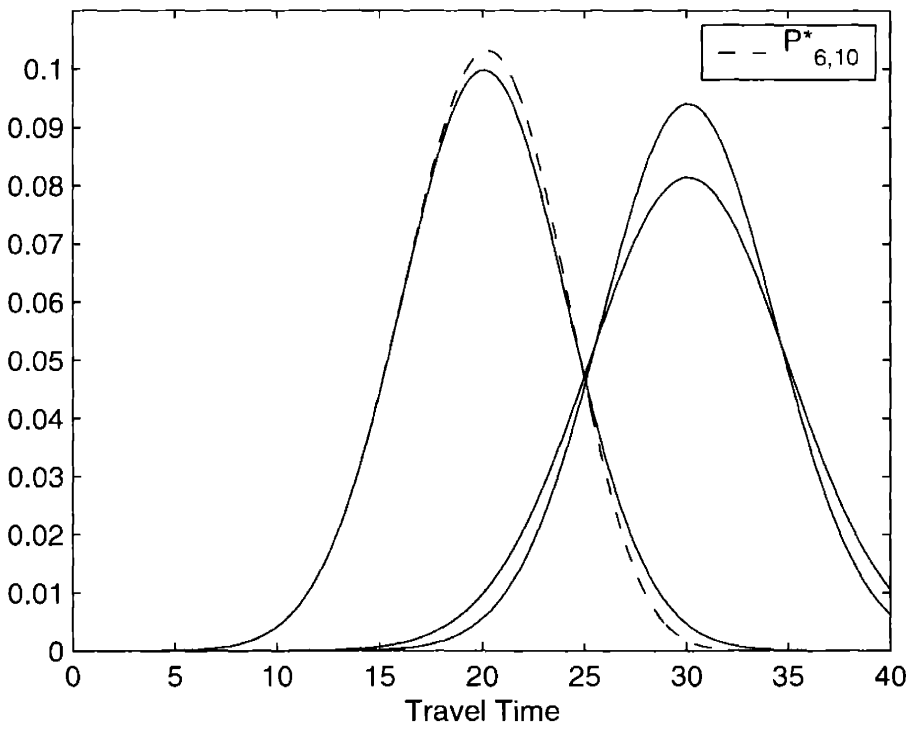


Figure 6-15: Comparison of travel time density functions from node 6 to node 10. The dashed curve is the density function of the minimum travel time distribution $P^*_{6,10}$. The three remaining curves correspond to the travel time density functions of path 6-5-9-8-10 and path 6-5-8-10 ($\mathcal{N}(30.1, 24)$), path 6-5-7-8-10 ($\mathcal{N}(30.1, 18)$), and path 6-9-8-10 ($\mathcal{N}(20.1, 16)$).

this represents a 0.1% decrease in expectation and a 76.1% decrease in the variance of the actual travel time.

Figure 6-17(a) shows the paths traversed when routing according to Equation 6.2. The associated actual realized travel times are presented in Figure 6-17(b). When $\theta = 0$, the sample mean of the actual travel time $\hat{E}[D_{2,10}]$ is 26.5448. The corresponding sample variance $\widehat{Var}(D_{2,10})$ is 14.8394. For $\theta = 2$, the sample mean of the actual travel time $\hat{E}[D_{2,10}]$ is 29.5791. The corresponding sample variance $\widehat{Var}(D_{2,10})$ is 9.7600. Relative to the expectation and variance when $\theta = 0$, this represents a 11.4% increase in expectation and a 34.2% decrease in the variance of the actual travel time.

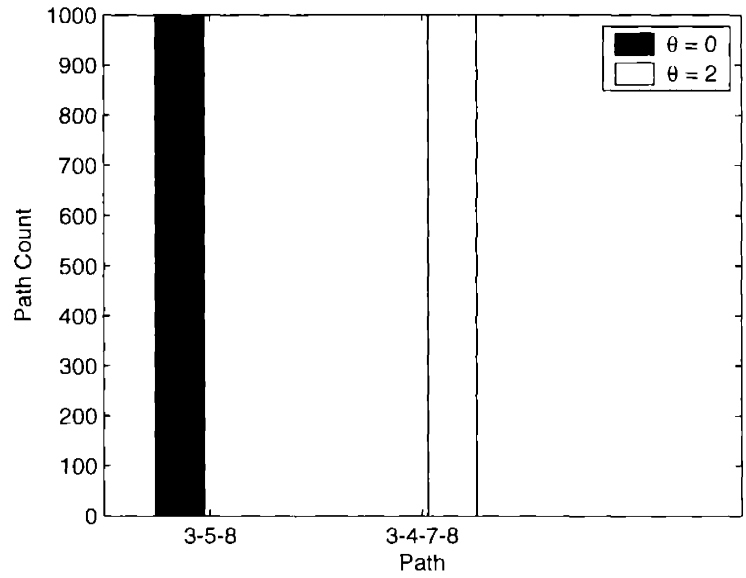
We see from these figures that when $\theta = 0$, multiple different paths are traversed. However, as soon as variance is factored into the routing objective (by setting $\theta = 2$), a single path is generally used (path 2-3-4-7-8-10 is path "2" in Figure 6-17(a)). Note the sample variance of the actual travel times is lower when $\theta = 2$ relative to when $\theta = 0$. However, the sample expected travel time is higher. Therefore, for a user who sets $\theta = 2$, the utility of low variance in travel time outweighs the higher expected travel time.

Comparing routing performance across routing objectives, we see that when routing according to Equation 6.2 and $\theta = 0$ the expected travel time decreased 11.6% relative to the expected travel time when routing according to Equation 6.1 and $\theta = 0$. There are almost no realizations larger than 45. Similarly, the variance of the actual travel time decreased by 35.9%. As in Section 6.2.1, the knowledge of the adjacent arc realizations appears to allow the user to decrease, in this case, both the expected travel time and the variance in travel time, even when the user is indifferent to the variance in travel time ($\theta = 0$). Therefore, when $\theta = 0$, there is value in knowing the adjacent arc travel time realizations.

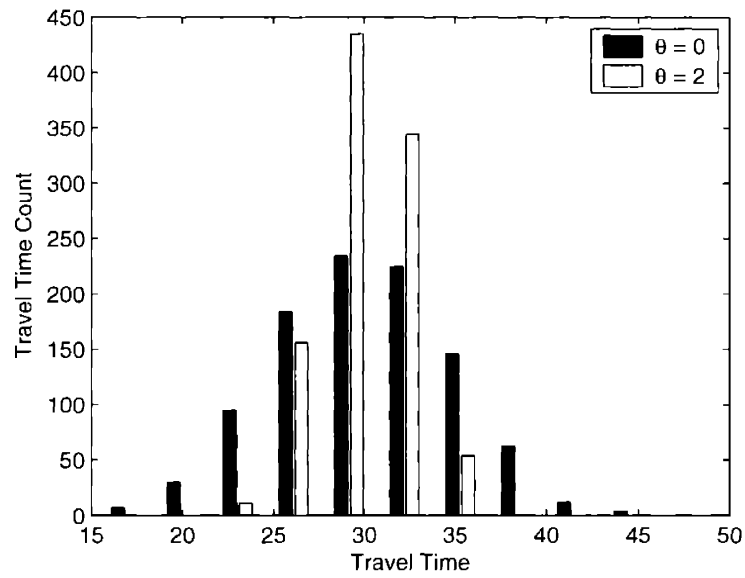
When $\theta = 2$, there is minimal change in the expected travel time (1.4% decrease) from Equation 6.1 to Equation 6.2. However, the travel time variance increases by 76.5%. Path 2-3-4-7-8-10 is the path with the lowest variance in travel time and is traversed the most when routing with Equation 6.2 and $\theta = 2$ (this path corresponds to path number 2 in Figure 6-17(a)). However, several other higher-variance paths are also traversed from node 2 to node 10, and this increases the overall variance in travel time. This is because some adjacent arc realizations are small enough that the disutility of high variance is offset. Although, in this example, the user obtains some utility for travel times with low variance, the user is also willing to accept higher variance in travel time when the adjacent arc realizations suggest a lower travel time. Note that, in this example, if the user is not willing to accept higher variance in travel time, the user must increase the disutility parameter θ appropriately.

The Performance of a "Confidence-Based" Routing Objective

As noted, the primary benefit of our routing procedure is that any routing objective can be calculated easily and efficiently. To emphasize this point, we consider a

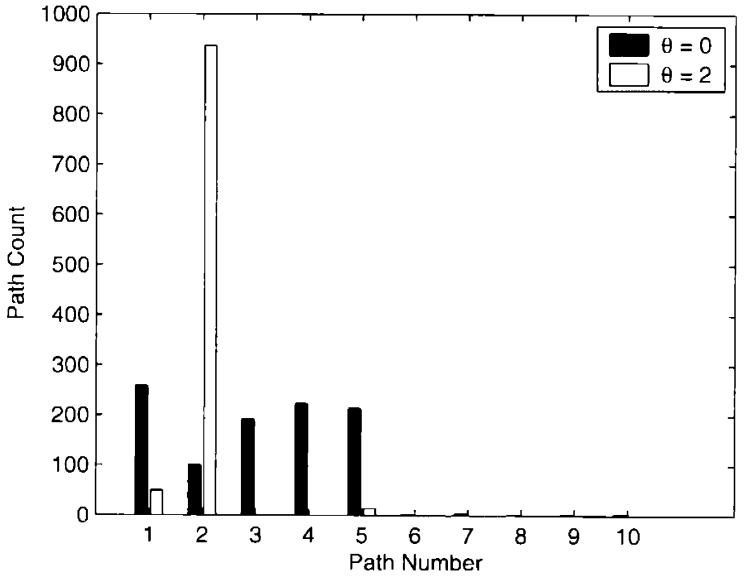


(a) Paths Traversed. Each path begins at node 2 and ends at node 10.

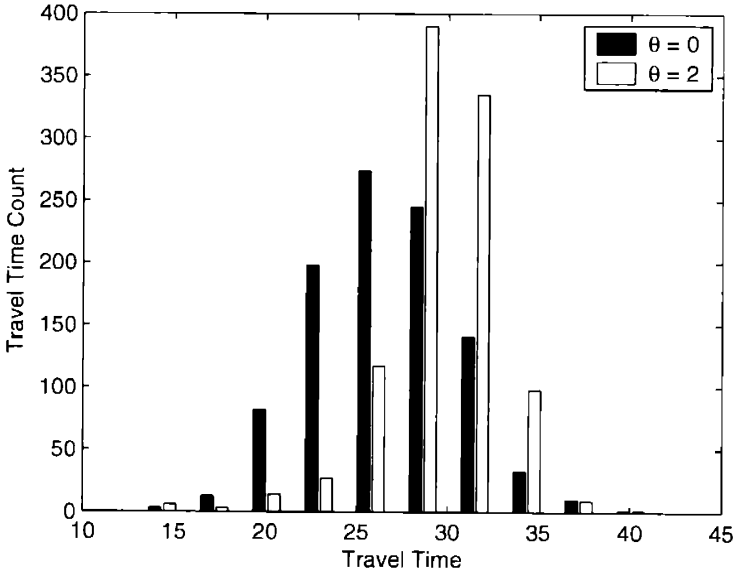


(b) Travel Times.

Figure 6-16: Routing performance from node 2 to node 10 on Figure 5-1 using the routing objective in Equation 6.1 with two different θs. Figure 6-16(a) displays the paths traversed and Figure 6-16(b) displays the corresponding travel times. For $\theta = 0, \hat{E}[D_{2,10}] = 30.0179$ and $\widehat{Var}(D_{2,10}) = 23.1403$. For $\theta = 2, \hat{E}[D_{2,10}] = 29.9877$ and $\widehat{Var}(D_{2,10}) = 5.5293$.



(a) Paths Traversed.



(b) Travel Times.

Figure 6-17: Routing performance from node 2 to node 10 on Figure 5-1 using the routing objective in Equation 6.2 with two different θs. Figure 6-17(a) displays the paths traversed and Figure 6-17(b) displays the corresponding travel times. For $\theta = 0, \hat{E}[D_{2,10}] = 26.5448$ and $\widehat{Var}(D_{2,10}) = 14.8394$. For $\theta = 2, \hat{E}[D_{2,10}] = 29.5791$ and $\widehat{Var}(D_{2,10}) = 9.7600$.

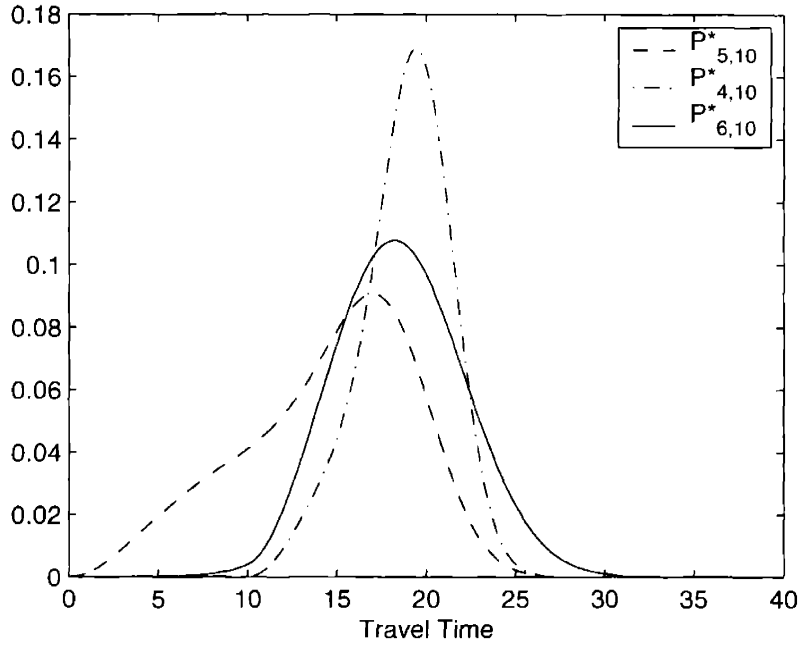


Figure 6-18: Comparison of the Density Functions of $P^*_{4,10}$, $P^*_{5,10}$, and $P^*_{6,10}$ with $X_{58} = \gamma(3, 6\frac{2}{3})$.

different type of routing objective that is described by the following equation:

$$j^* \in \underset{j \in N^+(i)}{\operatorname{argmin}}(-\Pr(X_{ij} + P^*_{jt} \leq \xi)) \tag{6.3}$$

This routing objective represents a certain confidence (expressed as a probability) that the travel time from node i to node t through node j with be less than or equal to ξ.

Consider the case of a user located at node 3 destined for node 10 on the network in Figure 5-1. Furthermore, for illustrative purposes, let $X_{58} = X_{85} = \gamma(3, 6\frac{2}{3})$ (a non-symmetric distribution)[1]. $\gamma(\alpha, \beta)$ denotes a gamma distribution with parameters α and β. The expectation of such a gamma distribution is $\alpha\beta$ or, in our case, $E[X_{58}] = 20$ as before. We now need to recalculate $P^*_{4,10}$, $P^*_{5,10}$, and $P^*_{6,10}$ since X_{58} is different from before. The new distributions are displayed in Figure 6-18. The effect of $X_{58} = \gamma(3, 6\frac{2}{3})$ is most prevalent on $P^*_{5,10}$.

If $\xi = 30$, we see that $\Pr(X_{ij} + P^*_{jt} \leq \xi)$ is the largest for $j = 5$[2]. In Figure 6-19(a) the path traversed with this routing objective is compared to the path traversed with the routing objective in Equation 6.1 with $\theta = 2$. As is evident, the actual path traversed changes from path 2-3-4-7-8-10, which has low variance in travel time, to path 2-3-5-8-10, which has the largest probability (confidence) of the travel time being

[1]This modification is made to emphasize the potential impact on performance of routing according to Equation 6.3

[2]This can be seen through inspection by adding X_{35} to $P^*_{4,10}$, $P^*_{5,10}$, and $P^*_{6,10}$ in Figure 6-18.

less than 30.

To appreciate the impact of routing with Equation 6.3, consider the corresponding realized travel times in Figure 6-19(b). The white bars correspond to Equation 6.1 with $\theta = 2$ and the black bars correspond to Equation 6.3. With Equation 6.1 the sample mean of the actual travel time $\hat{E}[D_{2,10}]$ is 29.9877. The sample mean of the actual time with Equation 6.3 is quite similar with $\hat{E}[D_{2,10}] = 29.8192$.

If a user were to make routing decisions solely on the expected travel time then there would be no preference for path 2-3-5-8-10 over path 2-3-4-7-8-10. However, the actual travel times are distributed quite differently over these two paths. By inspection, we see that there is low variance in the actual travel times with Equation 6.1 and $\theta = 2$. On the other hand, we observe that the actual travel times are positively skewed with Equation 6.3.

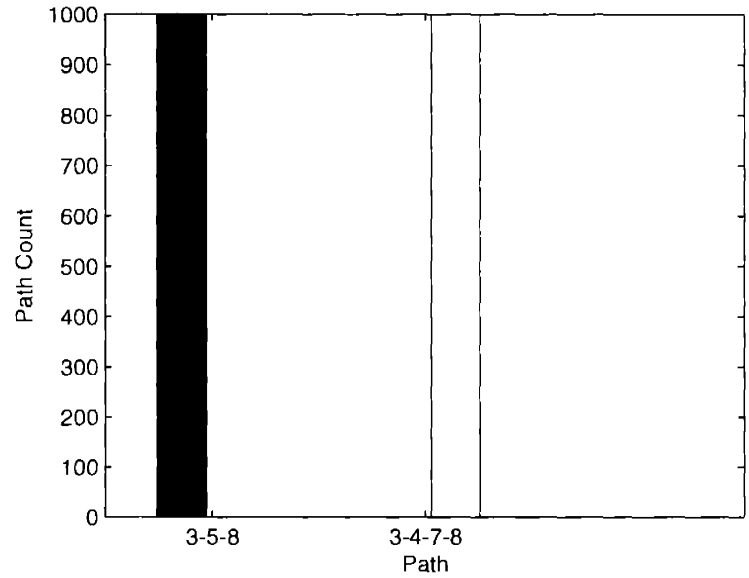
More formally, the sample variance of the actual travel times with Equation 6.1 (with $\theta = 2$) is $\widehat{Var}[D_{2,10}] = 5.5293$ and the sample variance of the actual travel times with Equation 6.3 is $\widehat{Var}[D_{2,10}] = 134.7661$. This agrees with the use of path 2-3-4-7-8-10 when routing with Equation 6.1 and $\theta = 2$. If we calculate the sample skewness of the actual travel times with Equation 6.1 (and $\theta = 2$) and Equation 6.3 we have -0.0441 and 0.9971 respectively. The positive skewness of the actual travel times along path 2-3-5-8-10 agrees with the selection of $j^* = 5$ at node 3.

As an additional note, in Chapter 2 we assume that each X_{ij} has the property that $\lim_{x_{ij} \to a^+} f_{X_{ij}}(x_{ij}) = 0$ and $\lim_{x_{ij} \to b^-} f_{X_{ij}}(x_{ij}) = 0$. If this assumption does not hold, the intermediate distributions calculated by Algorithm 4.3 and Algorithm 4.4 might not be continuous. Without the assumption of continuity, the interpolation routine could produce unexpected results. In Appendix A, we illustrate this problem further.

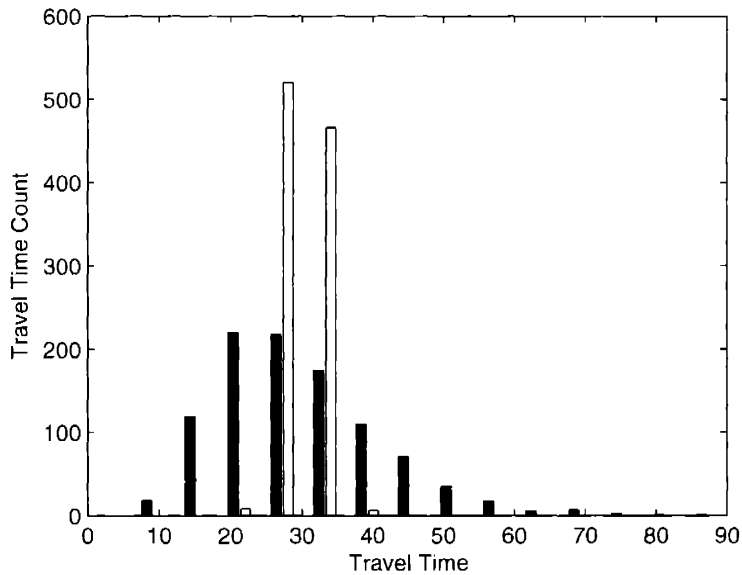
6.2.3 A Larger Network

The approach to routing presented in this thesis is computationally challenging. As such, it is important to understand performance in the context of larger and more practical networks than those discussed in Section 6.2.1 and Section 6.2.2. Consider the network in Figure 6-20. This network contains 522 arcs and 200 nodes. As noted, this is a modified model of the Amsterdam A-10 beltway. We modified the original model of the Amsterdam A-10 network so that arc $(j, i) \in A$ if arc $(i, j) \in A$. The purpose of this modification is to provide a check on the performance of our routing procedure. This is necessary because the network is sufficiently large and it is not trivial to manually confirm the results. By adding arc (j, i) to A if arc $(i, j) \in A$, the actual travel times from a source s to a sink t should be similar to the actual travel times when t is the *source* and s is the *sink*. For completeness, this property is confirmed in Appendix C. Four additional source and sink nodes were also added to the original model of the Amsterdam A-10 beltway. Appendix B contains the complete network.

We omit discussion of the calculation of any minimum travel time distribution P^*_{st} as this network is large and complex enough such that any discussion would provide



(a) Paths Traversed. Each path begins at node 2 and ends at node 10.



(b) Travel Times.

Figure 6-19: Comparison of routing performance between Equation 6.1 ($\theta = 2$) and Equation 6.3 from node 2 to node 10 on Figure 5-1. The black bars correspond to Equation 6.3. With Equation 6.1, $\hat{E}[D_{2,10}]$ is 29.9877. With Equation 6.3, $\hat{E}[D_{2,10}] = 29.8192$. With Equation 6.1, $\widehat{Var}[D_{2,10}] = 5.5293$. With Equation 6.3 $\widehat{Var}[D_{2,10}] = 134.7661$.

little insight. The examples in Section 6.2.1 and Section 6.2.2 are much better suited for gaining insight into our routing procedure. To gain a feel for the $\mathcal{TT}$-$\mathcal{DAG}$s generated by Algorithm 5.3 on this network, Figure 6-21 presents the $\mathcal{TT}$-$\mathcal{DAG}$ G' generated from node 144 to node 217. For the rest of this section, we focus on the performance of our routing procedure.

In the simulation, there were four source-sink pairs: node 214 to node 215, node 216 to node 217, node 215 to node 214, and node 217 to node 216. The simulation ran ten times for each value of θ; each run contained a different seed for the random number generator. In each run, 100 users traveled between each source-sink pair (400 users total in the network for each run). Therefore, over ten runs, each source-sink pair had a total of 1000 users. The routing objectives in Equation 6.1 and Equation 6.2 were used. For a particular routing objective and a particular random number generator seed, the simulation took approximately 40 minutes. This includes the calculation of all the necessary minimum travel time distributions and the generation of the necessary $\mathcal{TT}$-$\mathcal{DAG}$s. The simulation does not calculate any minimum travel time distributions or $\mathcal{TT}$-$\mathcal{DAG}$s for nodes that are not visited by a user. When possible, the minimum travel time distributions were cached so as to avoid repeating the same calculation. On the other hand, the $\mathcal{TT}$-$\mathcal{DAG}$s were not cached as they are computed much more efficiently than the minimum travel time distributions.

Figure 6-22 and Figure 6-23 show the results of routing from node 214 to node 215 with Equation 6.1 and Equation 6.2 respectively.

When routing according to Equation 6.1 and $\theta = 0$, the sample mean of the actual travel time $\hat{E}[D_{214,215}]$ is 49.9732. The corresponding sample variance $\widehat{Var}(D_{214,215})$ is 21.6779. For $\theta = 2$, the sample mean of the actual travel time $\hat{E}[D_{214,215}]$ is 55.0334. The corresponding sample variance $\widehat{Var}(D_{214,215})$ is 13.4035. Relative to the expectation and variance when $\theta = 0$, this represents a 10.1% increase in expectation and a 38.2% decrease in the variance of the actual travel time.

For Equation 6.2 and $\theta = 0$, the sample mean of the actual travel time $\hat{E}[D_{214,215}]$ is 50.0032. The corresponding sample variance $\widehat{Var}(D_{214,215})$ is 21.6349. For $\theta = 2$, the sample mean of the actual travel time $\hat{E}[D_{214,215}]$ is 53.9420. The corresponding sample variance $\widehat{Var}(D_{214,215})$ is 19.0718. Relative to the expectation and variance when $\theta = 0$, this represents a 7.9% increase in expectation and a 11.8% decrease in the variance of the actual travel time.

Comparing routing performance across routing objectives, we see that when routing with Equation 6.2 and $\theta = 0$ the expected travel time increased 0.06% relative to the expected travel time when routing with Equation 6.1 and $\theta = 0$. The variance of the actual travel time decreased by 0.2%. If we examine the paths traversed when $\theta = 0$, we see that path number 2 is used almost exclusively regardless of whether Equation 6.1 or Equation 6.2 is used as the routing objective. Path number 2 is so dominant in expected travel time that almost no other other path is traversed even though the adjacent arc realizations are known.

When $\theta = 2$, there is minimal change in the expected travel time (2.0% decrease) from Equation 6.1 to Equation 6.2. However, the travel time variance increases by 42.3%. These results are analogous to the results in comparing Equation 6.1 to

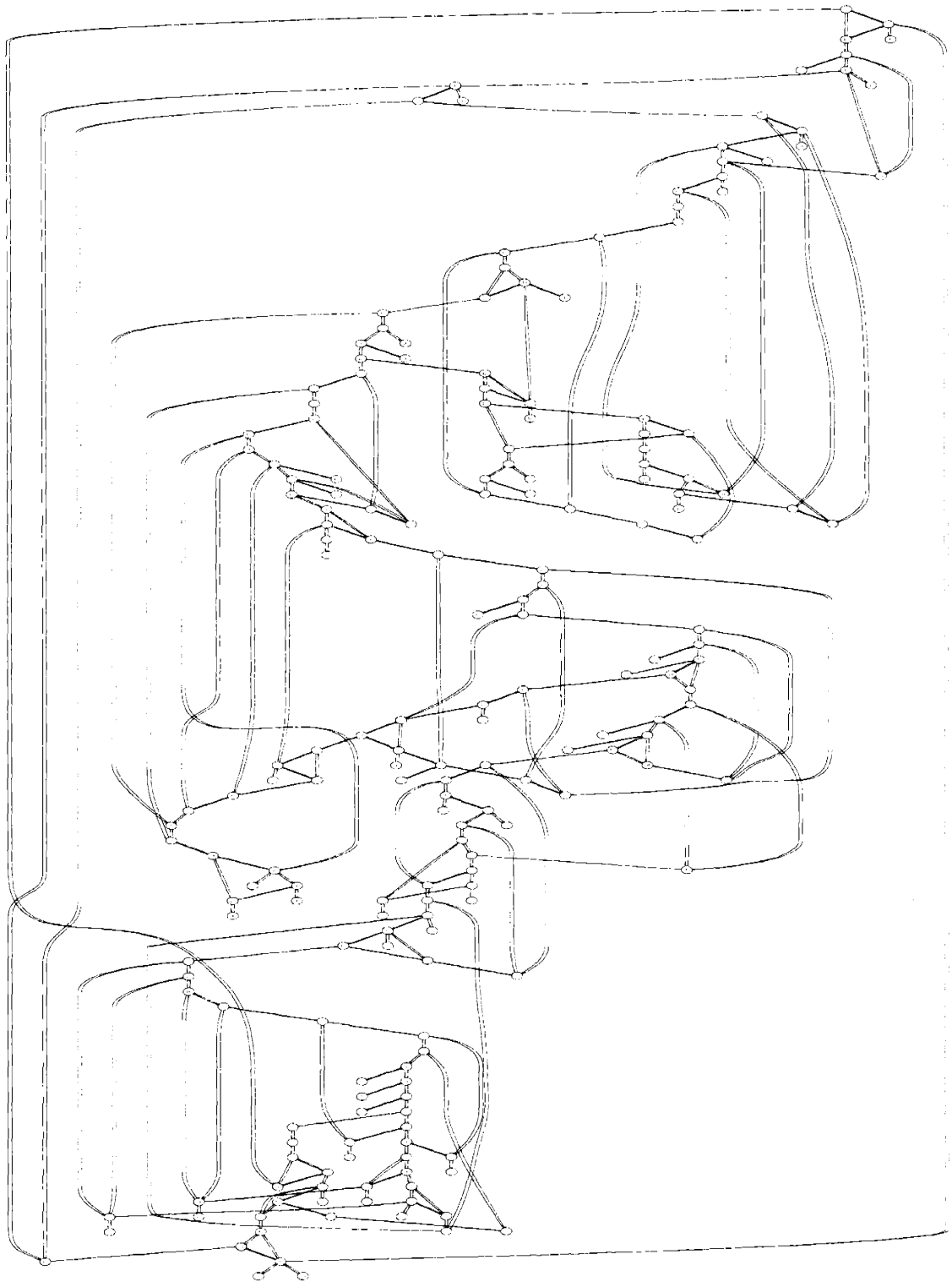


Figure 6-20: A larger network. This a modified model of the Amsterdam A-10 beltway and contains 522 arcs and 200 nodes.

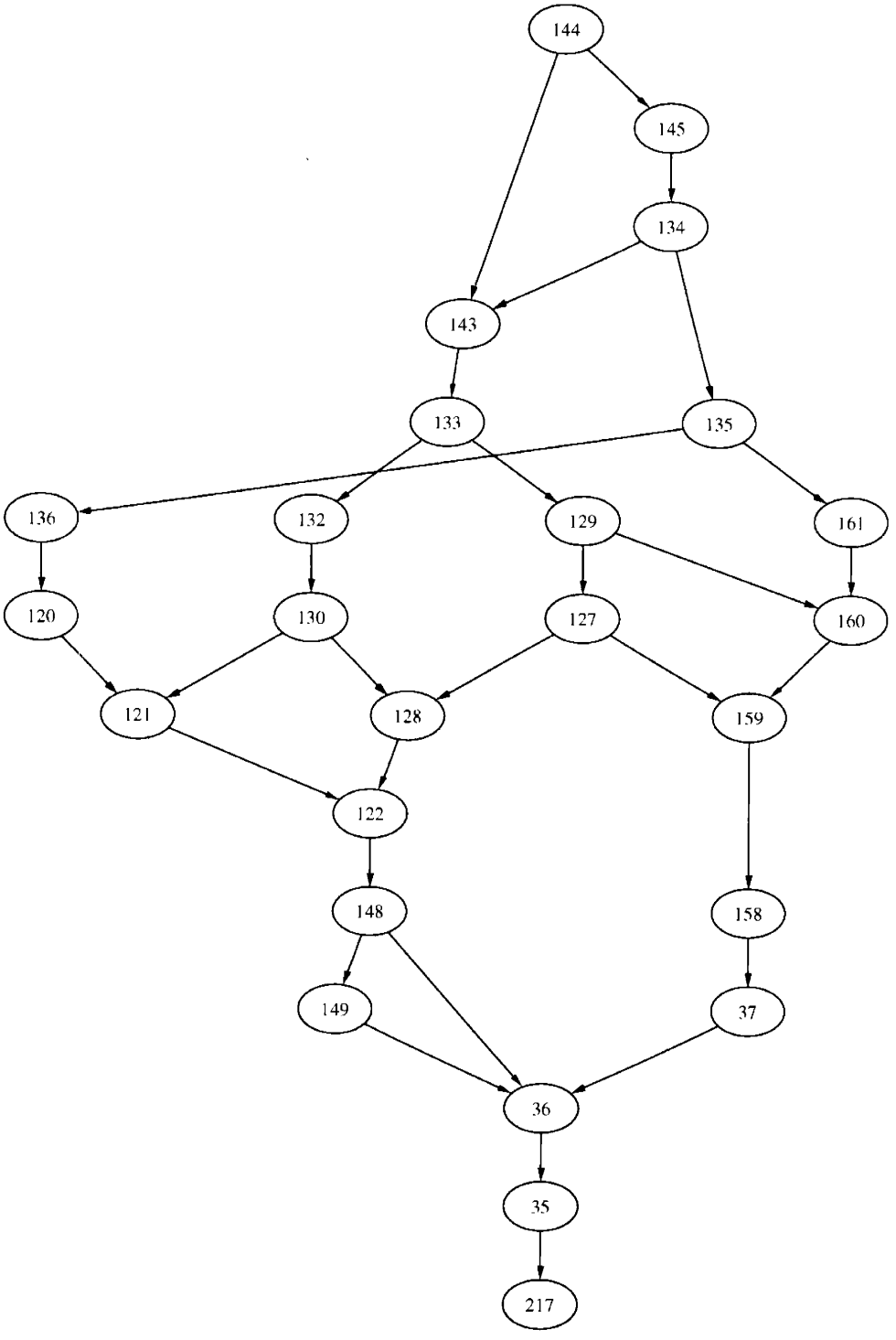


Figure 6-21: The Subgraph Generated by Algorithm 5.3 on Figure 6-20 from Node 144 to Node 217.

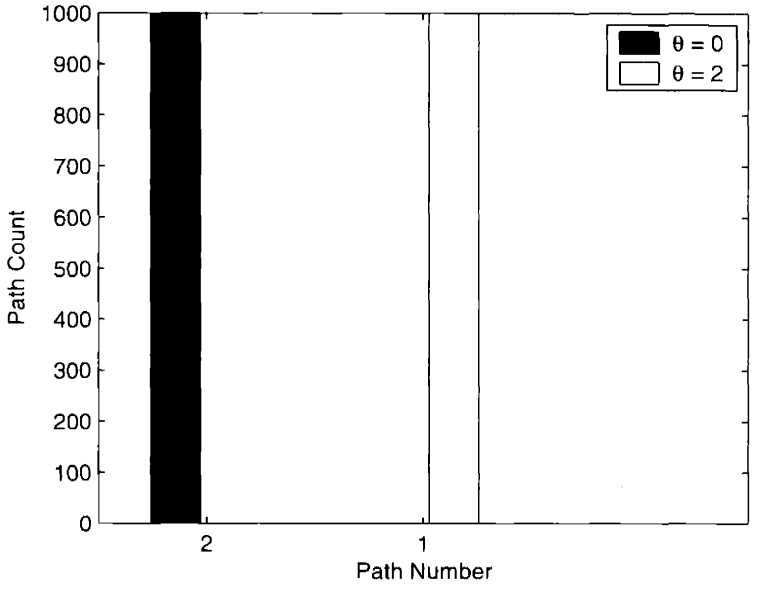
Equation 6.2 in Section 6.2.2, and the same analysis follows.

Figure 6-24 and Figure 6-25 show the results of routing from node 216 to node 217 with Equation 6.1 and Equation 6.2 respectively.

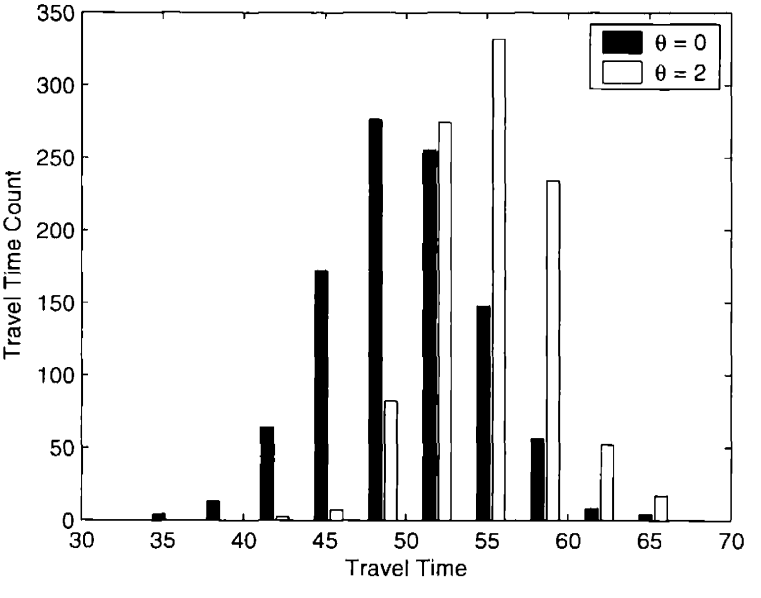
When routing according to Equation 6.1 and and $\theta = 0$, the sample mean of the actual travel time $\hat{E}[D_{216,217}]$ is 80.1845. The corresponding sample variance $\widehat{Var}(D_{216,217})$ is 15.0585. For $\theta = 2$, the sample mean of the actual travel time $\hat{E}[D_{216,217}]$ is 80.0151. The corresponding sample variance $\widehat{Var}(D_{216,217})$ is 16.2935. Unlike the users traveling from node 214 to node 215, the same path is traversed regardless of whether $\theta = 0$ or $\theta = 2$ in Equation 6.1. In this case, path number 2 has both a low expected travel time and low variance in travel time. Therefore, in this case, the user disutility for high variance is inconsequential in the result of the routing decision.

When routing according to Equation 6.2 and $\theta = 0$, the sample mean of the actual travel time $\hat{E}[D_{216,217}]$ is 78.2601. The corresponding sample variance $\widehat{Var}(D_{216,217})$ is 14.1328. For $\theta = 2$, the sample mean of the actual travel time $\hat{E}[D_{216,217}]$ is 78.5087. The corresponding sample variance $\widehat{Var}(D_{216,217})$ is 14.7434. Relative to the expectation and variance when $\theta = 0$, this represents a 0.3% increase in expectation and a 4.3% increase in the variance of the actual travel time. We see from Figure 6-25(a) that the distribution of paths traversed is relatively similar for both $\theta = 0$ and $\theta = 2$.

If we compare the routing performance across routing objectives, we see that when routing with Equation 6.2 and $\theta = 0$ the expected travel time decreased 2.4% relative to the expected travel time when routing with Equation 6.1 and $\theta = 0$. The variance of the actual travel time decreased by 6.1%. Though the sample expected travel time and sample variance slightly decreased, the real value of knowing the adjacent arc travel time realizations is that the actual travel times are distributed similarly to Figure 6-24(b), but over a greater range of paths. Such a property might be useful to distribute the load in a network subject to congestion. Similar results and observations follow for $\theta = 2$: the sample expected travel time decreases 1.9% and travel time variance decreases by 9.5% from Equation 6.1 to Equation 6.2.

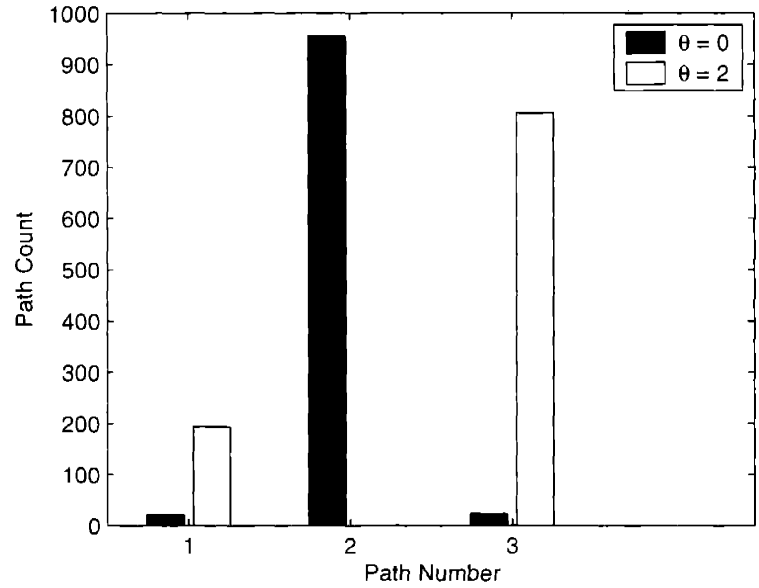


(a) Paths Traversed.

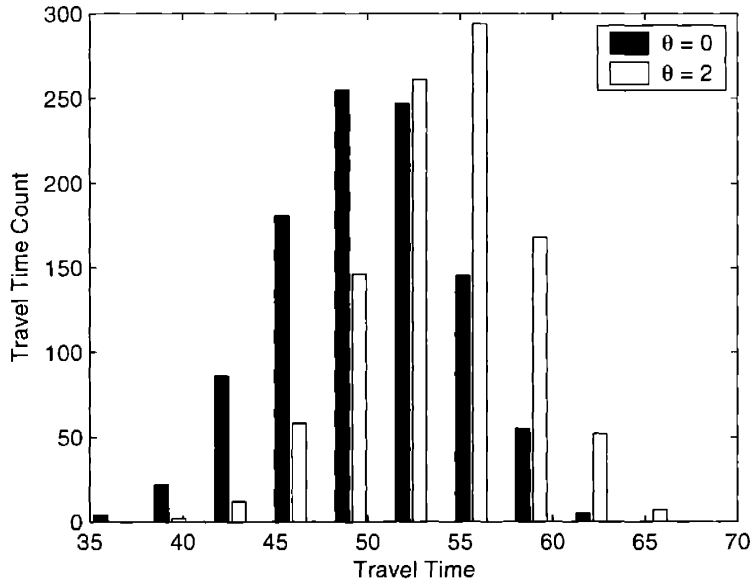


(b) Travel Times.

Figure 6-22: Routing performance from node 214 to node 215 on Figure 6-20 using the routing objective in Equation 6.1 with two different θs. Figure 6-22(a) displays the paths traversed and Figure 6-22(b) displays the corresponding travel times. For $\theta = 0, \hat{E}[D_{214,215}] = 49.9732$ and $\widehat{Var}(D_{214,215}) = 21.6779$. For $\theta = 2, \hat{E}[D_{214,215}] = 55.0334$ and $\widehat{Var}(D_{214,215}) = 13.403$.

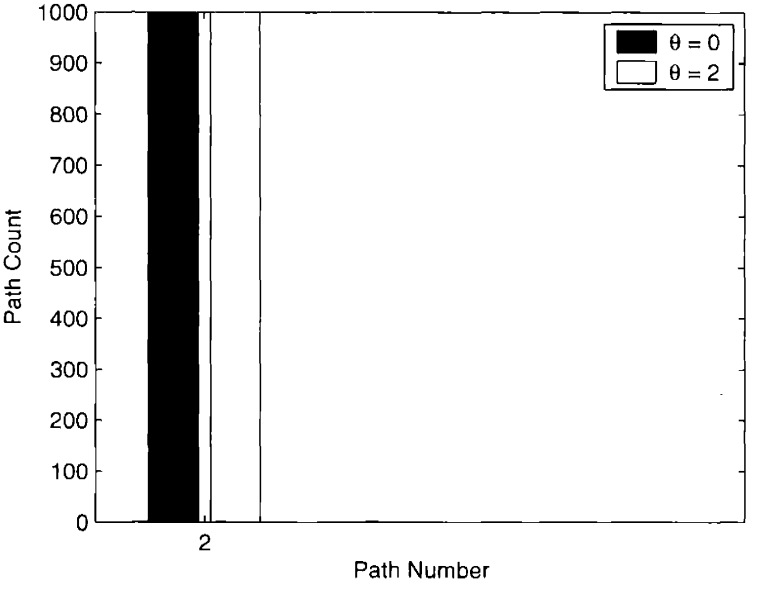


(a) Paths Traversed.

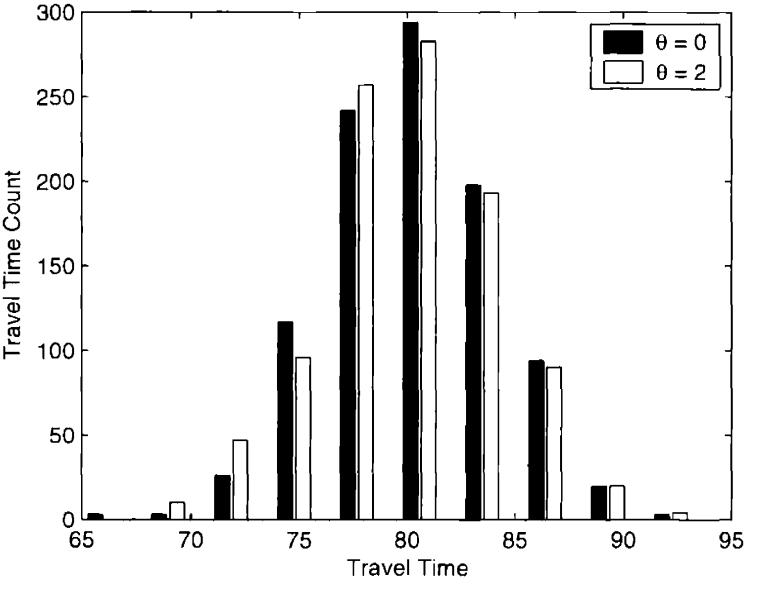


(b) Travel Times.

Figure 6-23: Routing performance from node 214 to node 215 on Figure 6-20 using the routing objective in Equation 6.2 with two different θs. Figure 6-23(a) displays the paths traversed and Figure 6-23(b) displays the corresponding travel times. For $\theta = 0, \hat{E}[D_{214,215}] = 50.0032$ and $\widehat{Var}(D_{214,215}) = 21.6349$. For $\theta = 2, \hat{E}[D_{214,215}] = 53.9420$ and $\widehat{Var}(D_{214,215}) = 19.0718$.

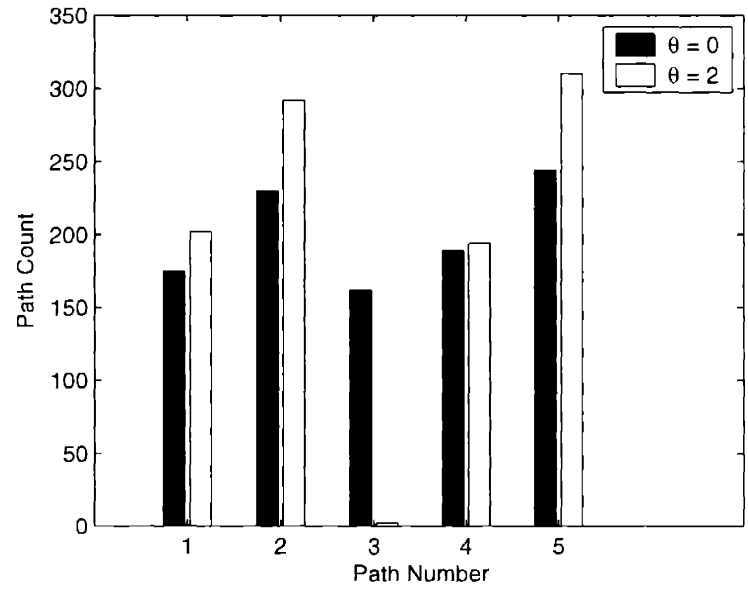


(a) Paths Traversed.

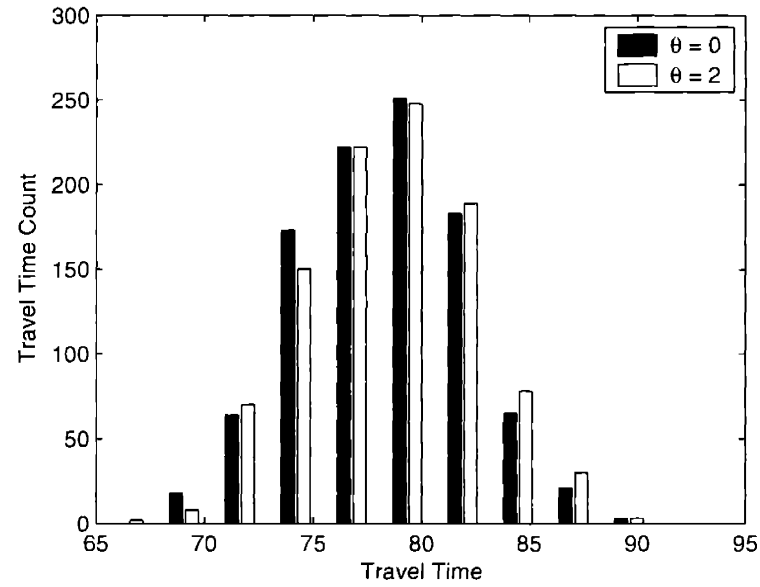


(b) Travel Times.

Figure 6-24: Routing performance from node 216 to node 217 on Figure 6-20 using the routing objective in Equation 6.1 with two different θs. Figure 6-24(a) displays the paths traversed and Figure 6-24(b) displays the corresponding travel times. For $\theta = 0$, $\hat{E}[D_{216,217}] = 80.1845$ and $\widehat{Var}(D_{216,217}) = 15.0585$. For $\theta = 2$, $\hat{E}[D_{216,217}] = 80.0151$ and $\widehat{Var}(D_{216,217}) = 16.2935$.



(a) Paths Traversed.



(b) Travel Times.

Figure 6-25: Routing performance from node 216 to node 217 on Figure 6-20 using the routing objective in Equation 6.2 with two different θs. Figure 6-25(a) displays the paths traversed and Figure 6-25(b) displays the corresponding travel times. For $\theta = 0, \hat{E}[D_{216,217}] = 78.2601$ and $\widehat{Var}(D_{216,217}) = 14.1328$. For $\theta = 2, \hat{E}[D_{216,217}] = 78.5087$ and $\widehat{Var}(D_{216,217}) = 14.7434$.

Chapter 7

Conclusions and Future Research

In this chapter, we provide a summary of the work in this thesis. We consider the results and the limitations of this thesis. We also discuss areas of future research.

7.1 Summary

Travel times in certain types of networks are inherently probabilistic . Transportation and data networks are examples of such networks. Even with advanced technologies and information systems, travel time is likely to be predictable at best with uncertainty. For example, in the case of transportation networks, there will always be random factors affecting travel time in the form of incidents, road conditions, and weather.

Under probabilistic conditions, routing systems typically focus on achieving the least expected travel time. Such an objective is natural, but one of the problems with this approach is that it does not take into account the fact that a user might prefer routing options with better attributes such as a *higher* expected travel time in exchange for a *lower* variance in travel time. A second problem is that such an approach implicitly assumes each user in the network has the same routing objective. Consequently, in probabilistic networks, a routing system that addresses these problems will further advance the quality of routing.

In this thesis we develop an approach to routing in probabilistic networks that addresses these problems. We address the inherent difficulty of routing in probabilistic networks by proposing a routing procedure that quantifies the quality of a routing option, and allows for different user routing objectives. More formally, the fundamental concept in this thesis is, for a given user with a set of routing options, we approximate the distribution of travel time for each routing option. We are motivated by the observation that *if* we knew the travel time distribution for each routing option, we could determine the best routing option for *any* user routing objective. Furthermore, user routing objectives would need not be specified *a priori*, but could be made adaptively as the user travels through the network.

The routing system we propose is presented in Chapter 2. This system is executed every time a user arrives at a node i destined for a node t. Rather than selecting

an entire path for the user, the system only determines the next "neighbor" node j^*, $j^* \in N^+(i)$, for the user to travel to. Ideally, we would know the actual travel time distribution, D_{jt}, from each node $j \in N^+(i)$ to sink t. D_{jt} is the distribution of travel time over the path traversed from node i to node t. However, by definition, D_{jt} only becomes known *a posteriori*. In light of this, we choose to approximate D_{jt} with P^*_{jt}, which is the minimum travel time distribution from node j to node t.

For each $j \in N^+(i)$, we initially generate an acyclic subgraph G' of G from node j to sink t since we do not assume the underlying network G is acyclic. Next, we determine whether G' is series-parallel or non-series-parallel. If G' is series-parallel, we calculate P^*_{jt} using an algorithm taken primarily from [26] and [43]. If G' is non-series-parallel, we approximate P^*_{jt} by using an algorithm based on [17].

Once each P^*_{jt} has been calculated, the selection of $j^* \in N^+(i)$ is done by a straightforward comparison. User routing objectives are specified in the form of a general routing objective, represented by the real-valued operator Γ, with two other metrics, Φ and Ψ, as arguments. Φ operates on arc travel time random variables (X_{ij}) and Ψ operates on minimum travel time distributions (P^*_{jt}). On-line or updated information can be incorporated into the objective by modifying X_{ij} or P^*_{jt} appropriately. For example, if the travel time x_{ij} of arc (i,j) is known with some certainty, Φ can be set to x_{ij}. Taken together, Γ, Φ, and Ψ are used to determine a single real-value that represents each $j \in N^+(i)$.

The primary benefit of this approach to routing is that, by computing adequate travel time distributions at nodes, we are able to improve the quality of routing decisions. We need not assume a particular routing objective. Additionally, once a routing objective is selected, the user is not bound to that routing objective and can switch objectives *en route*. We discussed several different routing objectives. Computational results demonstrate that we are able to make effective routing decisions with routing objectives that are based on the least expected travel time, a hybrid of expected travel time and travel time variance, and the probability of the travel time being less than a certain value. We also show how to incorporate adjacent arc travel time realizations into these objectives. However, the primary significance of this approach is that we are not limited to a single routing objective. In essence, under this routing system, the set of possible routing objectives is infinite.

On the other hand, there are limitations to our approach as the quality of the routing depends on the quality of P^*_{jt}, which depends on the computation of an acyclic graph G' from a possibly cyclic graph G, and on whether G' is a non-series-parallel network. We are also limited to approximating P^*_{jt} on non-series-parallel networks. We would prefer to be able to calculate P^*_{jt} exactly on non-series-parallel networks, but such a calculation quickly becomes difficult and inefficient.

7.2 Future Research

In terms of future research, one useful property of our approach to routing is that the components of our routing procedure are modular. That is, we can replace each component without affecting the other steps of the routing procedure. For example,

if we decide there is a better algorithm to calculate P^*_{jt} then we can just replace our current algorithm for calculating P^*_{jt} with the new algorithm. This modularity is useful because each component can be studied, improved, and replaced independently.

7.2.1 Dependence of Arc Travel Time Distributions

In this thesis, we assume that the arc travel time random variables X_{ij} are independent and known for each arc $(i, j) \in A$. The primary problem with this assumption is that the arc travel times may be dependent. With dependence among arc travel times, the sum and minimum functions are not as easily analyzed. Efficient methods and approaches that consider partial or full arc travel time dependence would be beneficial.

One approach to dependence that could easily be incorporated into our routing system is the notion of conditional independence. When a user that originates at node s arrives at node i, the travel times of arcs that have not yet been traversed could be conditioned on the travel time realizations of the arcs used from node s to node i. Similarly, in a network subject to congestion, the travel times of arcs that have not yet been traversed could be conditioned on the levels of congestion of the traversed arcs. If the arcs not yet traversed are conditionally independent given the travel times of the traversed arcs, then we can *still* use the methods in this thesis to calculate P^*_{jt}. In the implementation, this would amount to a table lookup of the conditioned arc travel time distributions before calculating P^*_{jt}. The notion of conditional independence within a transportation network has some similarities to *Bayesian networks* [37].

In some sense, we have already addressed a form of conditioning in this thesis. In Equation 6.2, we assume the travel times on the adjacent arcs are known immediately before the arc is traversed. By setting $\Phi(X_{ij})$ equal to the known travel time x_{ij}, we effectively shift the minimum travel time distribution P^*_{jt} by x_{ij} to obtain a better approximation of the actual travel time distribution.

7.2.2 Estimation of Arc Travel Time Distributions

Another area of possible research is the initial estimation of the arc travel time distributions. In this thesis, we assume the arc travel time distributions are known *a priori*. However, they could also be generated from historical data or using sophisticated prediction models. Additionally, the densities could be estimated *on-line* using updated travel time information (obtained through an information system). Both [19] and [40] cover density estimation.

7.2.3 Computation of Minimum Travel Time Distributions

In this thesis, we approximate P^*_{jt} when the underlying graph G' is not series-parallel. This is achieved by conditioning on certain arcs and reducing G' to a series-parallel graph. In the case when we have the option of conditioning on different arcs, it would be useful to study the consequences of intelligently selecting the arc to condition on. In this thesis, arcs were chosen arbitrarily.

Additionally, in [1], Bein et al., define $s \rightarrow t$ $\mathcal{DAG}$ *complexity*, which is a measure of how "series-parallel" a $\mathcal{TT}$-$\mathcal{DAG}$ is. This measure is defined as the minimum number of nodes (and adjacent arcs) that need to be removed to make G series-parallel reducible. While Bein et al. develop an efficient algorithm to find the $s \rightarrow t$ $\mathcal{DAG}$ complexity, it would also be useful to bound the $s \rightarrow t$ $\mathcal{DAG}$ complexity relative to the size of G'.

7.2.4 Time-Dependent Networks

There is a growing body of research in routing on *time-dependent* probabilistic networks (see [28], [27], [16], [22]). A time-dependent probabilistic network is a probabilistic network where the arc travel time distributions change with time. It would be useful to consider the benefits and the problems of applying this thesis to time-dependent probabilistic networks.

7.2.5 En Route Routing and Real-Time Information

The primary benefit of using travel time distributions is that, once the travel time distributions are known, the calculation of *any* routing objective becomes straightforward. One useful consequence of this is that a user is not bound to a particular routing objective for the duration of a trip. The user-specified routing objective can be changed *en route* with no additional computational overhead. For example, consider a user traveling from location s to location t for an important meeting. At the outset of the trip, the user might have the objective of arriving in the least expected time. Suppose, however, that along the way there are delays. In this case, due to the importance of the meeting, the user might want to change the objective to be the least *possible* travel time in order to have some hope of arriving at the meeting in time. Alternatively, the user might change the objective to compute the highest probability of arriving at the meeting in time.

In this thesis, we do not explicitly consider changing routing objectives *en route*, though it is already possible with our routing procedure. Some areas of study could include the identification of situations where the user would want to change the routing objective *en route*, and how the use of on-line information could affect the selection of the routing objective.

7.2.6 Numerical Analysis

We discuss numerical implementations of the sum and minimum of random variables. Though it is not the explicit focus of this thesis, efficient implementations of these functions are necessary for the calculation of travel time distributions. Fundamental to the implementation of these functions is the discrete representation of random variables. We represent random variables as arrays of the possible realizations and the corresponding densities. This representation was motivated algorithmically as it is convenient for use in the sum and minimum routines. Our results suggest this

representation also produces accurate results. Future research in this area would analyze this representation (or alternative representations) with numerical analysis.

The basic approach to computing the sum of random variables is to take the convolution of the corresponding probability density functions. Two basic convolution methods are direct convolution and Fast Fourier Transform convolution. These methods are well-defined and studied in a variety of fields. On the other hand, the efficient implementation of the minimum of random variables is not as well-studied, and deserves further research to develop a more efficient numerical implementation.

7.2.7 Practical Infrastructure Considerations

A favorable property of the routing system proposed is that it lends itself to be implemented as a decentralized infrastructure where the routing decisions are made at each node. In this case, each node needs to know the network topology and the arc travel time random variables of every arc in the network. Aside from this knowledge, each node can operate independently.

In implementations of the Bellman-Ford routing algorithm on computer networks, each node in the network maintains a *distance-vector*. The distance-vector of node i records the distance from each adjacent node $j \in N^+(i)$ to each destination in the network. In practice, the Bellman-Ford routing algorithm can be implemented in a decentralized manner where each node communicates its distance-vector to each of its adjacent nodes. Further research would modify this communications scheme to apply to our routing system.

Both [5] and [42] consider the practical benefits of decentralized computation. Three observations are emphasized here:

Scalability With a decentralized algorithm, it is easy to add a node to the underlying network. With a centralized algorithm there is need for configuration and coordination setup. This is the primary reason why computer networks employ a certain degree of decentralization. It is too hard to add a node to the network and maintain a centralized routing infrastructure.

Communications If a routing decision needs to be executed before the results of a centralized algorithm can be communicated, then such an algorithm is of no use. For probabilistic networks, it is imperative to consider the costs of data collection and calculation.

Single Point of Failure If a centralized implementation is disrupted, then so are all locations within the infrastructure. Decentralizing decision making allows for continuous network operation, even when a subset of the network has failed.

[This page intentionally left blank.]

Appendix A

Continuity and Minimum Travel Time Distributions

Consider the network in Figure 5-1 where X_{58} is set to $\mathcal{EXP}(20)$ (the exponential distribution with mean 20 and $\lambda = \frac{1}{20}$). $\mathcal{EXP}(20)$ is continuous over $[0, \infty)$, but $\lim_{x_{58} \to 0^+} f_{X_{58}}(x_{58}) \neq 0$. To understand the consequences of this, suppose X_{58} is shifted 10 units to the right. This might occur in **Step 2b** of Algorithm 4.4. With such a shift X_{58} is no longer defined for $x_{58} < 10$. Furthermore, this shift makes X_{58} non-continuous over $[0, \infty)$ as $\lim_{x \to 10^+} f_{X_{58}}(x_{58}) = 0.05$ and $\lim_{x \to 10^-} f_{X_{58}}(x_{58}) = 0$.

If we consider the case of a user located at node 3 going to node 10, the density functions of the resulting minimum travel time distributions $P^*_{5,10}$, $P^*_{4,10}$, and $P^*_{6,10}$ are shown in Figure A-1, Figure A-2, and Figure A-3 respectively. For P^*_{58}, X_{58} is not shifted, and $P^*_{5,10}$ is not affected by $X_{58} = \mathcal{EXP}(20)$. For $P^*_{4,10}$, X_{58} is shifted by $E[X_{45}] = 10$ since arc $(4, 5)$ is a Type-1 arc in Algorithm 4.4. This has the effect of making the density function of $P^*_{4,10}$ non-continuous at 10. The effects of a discontinuity at 10 are mitigated since any potential interpolation problems can be avoided by defining $P^*_{4,10}$ over $[a, b]$ where $a = 10$. This is not the case for $P^*_{6,10}$. Figure A-3 shows the density function of $P^*_{6,10}$ is non-continuous at 10. However, we cannot define an interval $[a, b]$ where the density function of $P^*_{6,10}$ is continuous. In this case, it is fortunate that the discontinuity is introduced in the final step of the calculation of $P^*_{6,10}$. In general, if a discontinuity is introduced during an intermediate step of Algorithm 4.4 (or Algorithm 4.3), the interpolation routine could produce unexpected results.

The purpose of the assumptions $\lim_{x_{ij} \to a^+} f_{X_{ij}}(x_{ij}) = 0$ and $\lim_{x_{ij} \to b^-} f_{X_{ij}}(x_{ij}) = 0$ is to prevent the introduction of a discontinuity that could corrupt the calculation of the minimum travel time distribution.

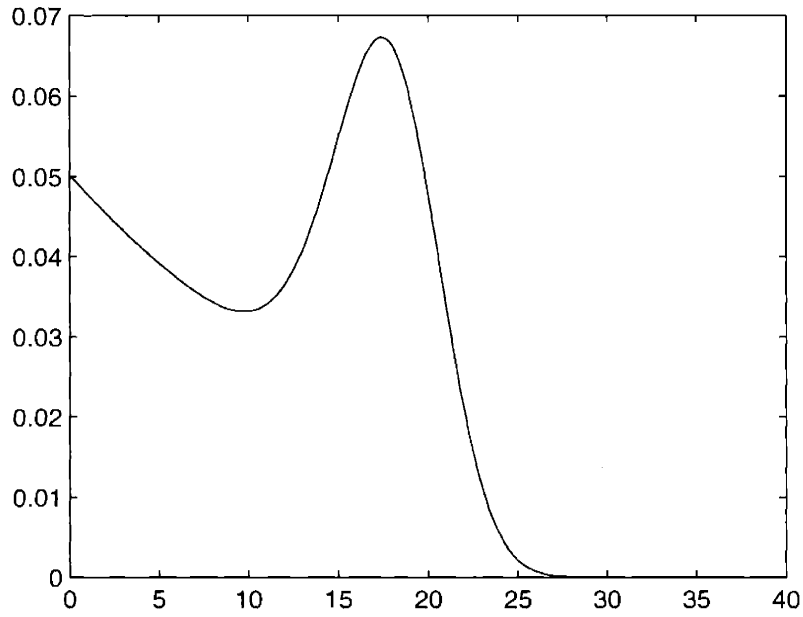


Figure A-1: The Density Function of $P^*_{5,10}$.

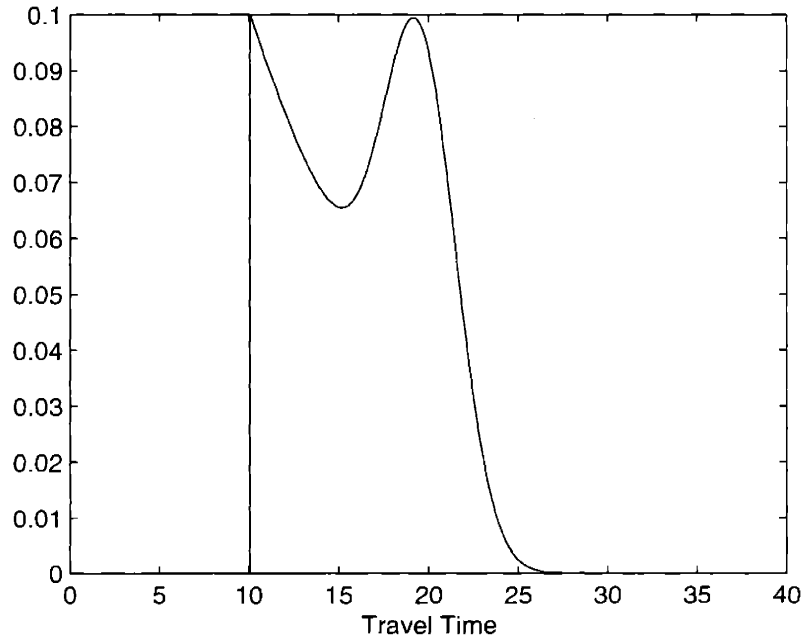


Figure A-2: The Density Function of $P^*_{4,10}$.

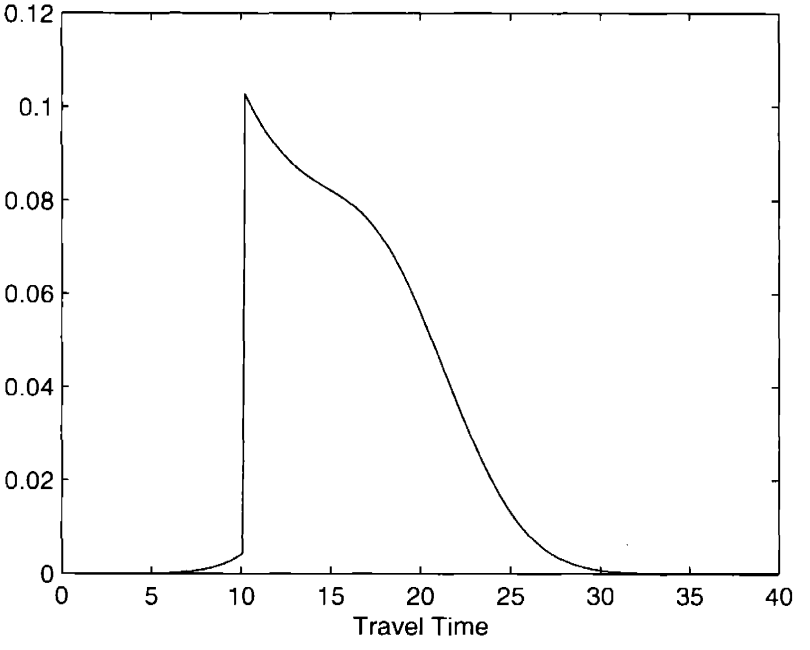


Figure A-3: The Density Function of $P^*_{6.10}$.

[This page intentionally left blank.]

Appendix B

Example Network in Figure 6-20

Head	Tail	Head	Tail	Head	Tail	Head	Tail	Head	Tail
101	48	48	101	213	201	201	213	174	49
49	174	49	175	175	49	51	52	52	51
47	48	48	47	48	49	49	48	49	50
50	49	43	44	44	43	44	45	45	44
45	46	46	45	103	45	45	103	44	173
173	44	172	44	44	172	45	104	104	45
52	53	53	52	99	51	51	99	51	100
100	51	176	52	52	176	52	177	177	52
55	57	57	55	58	56	56	58	56	54
54	56	98	57	57	98	56	178	178	56
67	68	68	67	68	69	69	68	68	91
91	68	54	211	211	54	57	54	54	57
90	68	68	90	189	69	69	189	69	190
190	69	84	82	82	84	82	81	81	82
82	192	192	82	210	80	80	210	80	86
86	80	88	89	89	88	89	77	77	89
89	86	86	89	79	84	84	79	87	78
78	87	83	80	80	83	83	87	87	83
212	83	83	212	86	85	85	86	84	87
87	84	165	167	167	165	167	5	5	167
5	6	6	5	6	168	168	6	168	16
16	168	17	170	170	17	17	9	9	17
105	16	16	105	18	11	11	18	11	106
106	11	105	11	11	105	19	17	17	19
16	15	15	16	12	14	14	12	168	169
169	168	169	12	12	169	12	171	171	12
169	170	170	169	170	171	171	170	12	11
11	12	105	106	106	105	106	9	9	106
9	107	107	9	20	107	107	20	171	172
172	171	172	173	173	172	173	174	174	173

Head	Tail	Head	Tail	Head	Tail	Head	Tail	Head	Tail
174	175	175	174	175	176	176	175	176	177
177	176	177	3	3	177	3	4	4	3
4	178	178	4	178	179	179	178	179	180
180	179	180	181	181	180	188	189	189	188
63	64	64	63	64	65	65	64	65	66
66	65	92	64	64	92	64	93	93	64
187	65	65	187	65	188	188	65	187	188
188	187	181	186	186	181	186	187	187	186
61	59	59	61	59	97	97	59	96	60
60	96	60	62	62	60	59	180	180	59
179	60	60	179	182	183	183	182	183	184
184	183	184	185	185	184	94	183	183	94
183	95	95	183	181	184	184	181	184	186
186	184	190	212	212	190	69	70	70	69
189	190	190	189	212	192	192	212	192	193
193	192	75	194	194	75	193	75	75	193
193	194	194	193	207	206	206	207	206	71
71	206	71	72	72	71	195	71	71	195
71	196	196	71	194	195	195	194	195	196
196	195	196	197	197	196	197	155	155	197
197	203	203	197	203	200	200	203	151	154
154	151	150	32	32	150	32	153	153	32
200	156	156	200	33	32	32	33	32	200
200	32	200	34	34	200	154	155	155	154
153	154	154	153	153	152	152	153	155	166
166	155	205	208	208	205	208	209	209	208
209	210	210	209	73	74	74	73	74	75
75	74	208	74	74	208	74	209	209	74
75	76	76	75	210	77	77	210	77	81
81	77	81	90	90	81	90	91	91	90
91	92	92	91	92	93	93	92	93	94
94	93	94	95	95	94	95	96	96	95
96	97	97	96	97	98	98	97	98	1
1	98	1	2	2	1	2	99	99	2
99	100	100	99	100	101	101	100	101	102
102	101	48	102	102	48	102	103	103	102
103	104	104	103	104	105	105	104	107	7
7	107	7	8	8	7	8	108	108	8
108	109	109	108	109	110	110	109	29	27
27	29	27	28	28	27	28	30	30	28
28	21	21	28	21	22	22	21	165	21
21	165	22	23	23	22	22	167	167	22
23	24	24	23	24	25	25	24	25	26
26	25	26	27	27	26	108	25	25	108

Head	Tail	Head	Tail	Head	Tail	Head	Tail	Head	Tail
26	109	109	26	110	111	111	110	163	114
114	163	112	113	113	112	113	114	114	113
114	115	115	114	114	164	164	114	110	113
113	110	113	111	111	113	111	119	119	111
119	120	120	119	161	162	162	161	139	140
140	139	142	139	139	142	140	141	141	140
162	139	139	162	140	119	119	140	163	164
164	163	164	165	165	164	120	121	121	120
123	128	128	123	128	127	127	128	127	126
126	127	159	127	127	159	127	129	129	127
129	160	160	129	129	133	133	129	133	143
143	133	143	144	144	143	144	145	145	144
145	134	134	145	134	143	143	134	134	135
135	134	135	161	161	135	121	130	130	121
132	130	130	132	130	128	128	130	128	122
122	128	144	147	147	144	120	136	136	120
145	146	146	145	135	136	136	135	136	131
131	136	131	132	132	131	132	133	133	132
121	122	122	121	160	161	161	160	122	148
148	122	158	159	159	158	159	160	160	159
162	163	163	162	148	149	149	148	149	150
150	149	150	151	151	150	152	202	202	152
151	152	152	151	203	156	156	203	156	157
157	156	157	37	37	157	37	158	158	37
148	36	36	148	36	149	149	36	35	36
36	35	36	37	37	36	37	38	38	37
157	158	158	157	201	203	203	201	201	202
202	201	202	204	204	202	204	205	205	204
204	206	206	204	206	205	205	206	214	49
49	214	215	103	103	215	216	143	143	216
217	35	35	217						

[This page intentionally left blank.]

Appendix C

Comparison of Actual Travel Times in the Example Network in Figure 6-20

In Section 6.2.3, we note that we modified the original model of the Amsterdam A-10 beltway to include arc (j, i) in A if arc $(i, j) \in A$. The purpose of this modification was to provide a check on the performance of our routing procedure. This is necessary because the network is sufficiently large and it is not trivial to manually confirm the results. By adding arc (j, i) to A if arc $(i, j) \in A$, the actual travel times from a source s to a sink t should be similar to the actual travel times when t is the *source* and s is the *sink*.

In this appendix, we confirm this property holds by inspection. Figure C-1 and Figure C-2 show the actual travel times when routing from source node 215 to sink node 214 with Equation 6.1 and Equation 6.2 respectively. Figure 6-22(b) and Figure 6-23(b) show the actual travel times when routing from source node 214 to sink node 215 with Equation 6.1 and Equation 6.2 respectively. As is evident, the distribution of actual travel times is similar.

Figure C-3 and Figure C-4 show the actual travel times when routing from source node 217 to sink node 216 with Equation 6.1 and Equation 6.2 respectively. Figure 6-24(b) and Figure 6-25(b) show the actual travel times when routing from source node 216 to sink node 217 with Equation 6.1 and Equation 6.2 respectively. As is evident, the distribution of actual travel times is similar.

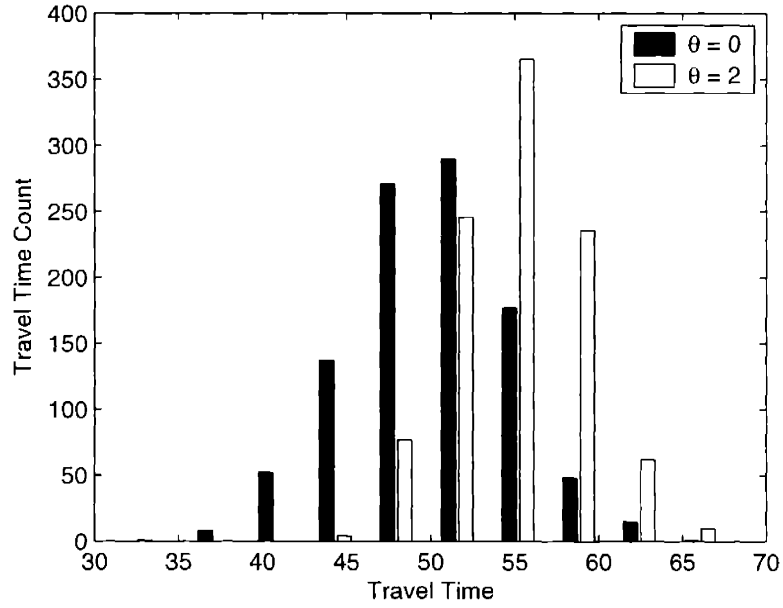


Figure C-1: Actual travel times from node 215 to node 214 on Figure 6-20 using the routing objective in Equation 6.1 with two different θs. For $\theta = 0$, $\hat{E}[D_{215,214}] = 50.1151$ and $\widehat{Var}(D_{215,214}) = 22.1952$. For $\theta = 2$, $\hat{E}[D_{215,214}] = 55.0833$ and $\widehat{Var}(D_{215,214}) = 14.0806$. These results agree with the results in Figure 6-22(b).

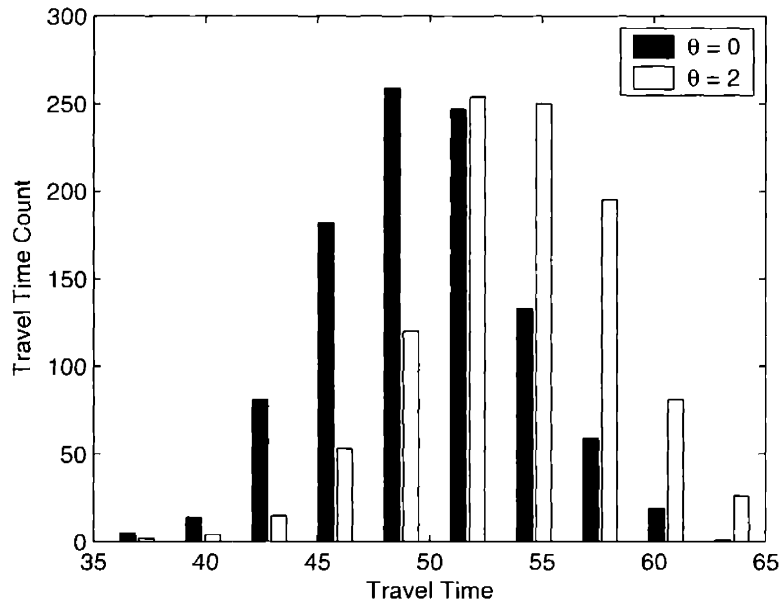


Figure C-2: Actual travel times from node 215 to node 214 on Figure 6-20 using the routing objective in Equation 6.2 with two different θs. For $\theta = 0$, $\hat{E}[D_{215,214}] = 49.8359$ and $\widehat{Var}(D_{215,214}) = 18.4830$. For $\theta = 2$, $\hat{E}[D_{215,214}] = 53.7591$ and $\widehat{Var}(D_{215,214}) = 19.3611$. These results agree with the results in Figure 6-23(b).

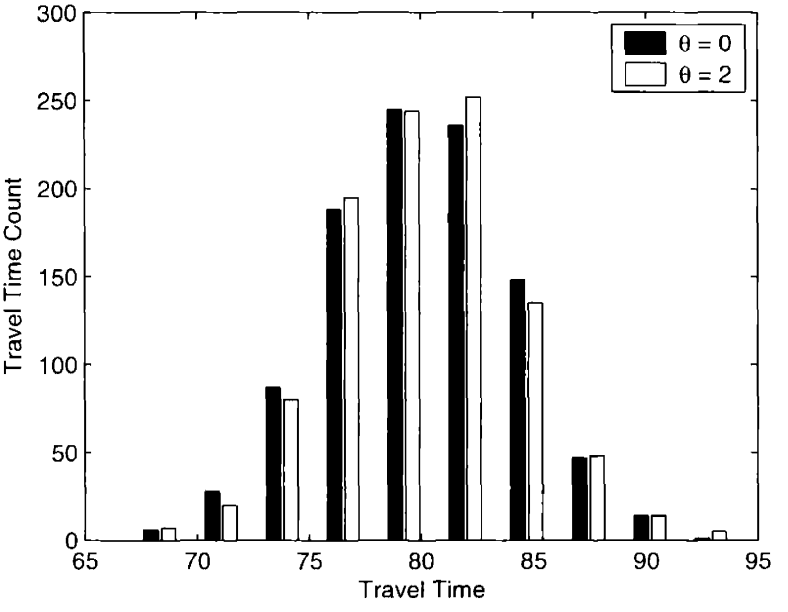


Figure C-3: Actual travel times from node 217 to node 216 on Figure 6-20 using the routing objective in Equation 6.1 with two different θs. For $\theta = 0$, $\hat{E}[D_{217,216}] = 79.9668$ and $\widehat{Var}(D_{217,216}) = 16.9835$. For $\theta = 2$, $\hat{E}[D_{217,216}] = 80.0642$ and $\widehat{Var}(D_{217,216}) = 16.5233$. These results agree with the results in Figure 6-24(b).

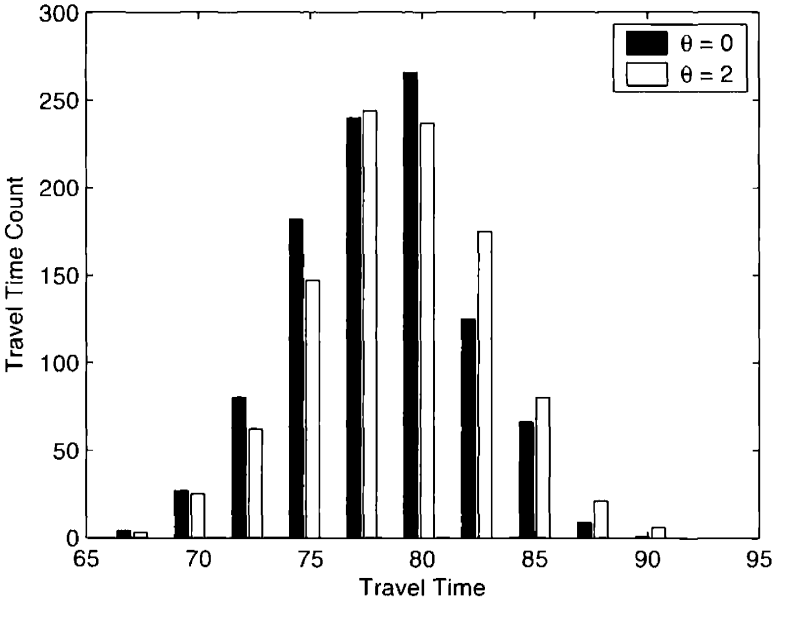


Figure C-4: Actual travel times from node 217 to node 216 on Figure 6-20 using the routing objective in Equation 6.2 with two different θs. For $\theta = 0$, $\hat{E}[D_{217,216}] = 78.1010$ and $\widehat{Var}(D_{217,216}) = 14.5568$. For $\theta = 2$, $\hat{E}[D_{217,216}] = 78.8317$ and $\widehat{Var}(D_{217,216}) = 15.6214$. These results agree with the results in Figure 6-25(b).

[This page intentionally left blank.]

Bibliography

[1] Wolfgang W. Bein, Jerzy Kamburowski, and Matthias F. M. Stallman. Optimal Reduction of Two-Terminal Directed Acyclic Graphs. *SIAM Journal on Computing*, 21:1112–1129, 1992.

[2] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Volume 1*. Athena Scientific, Belmont, MA, 2000.

[3] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Volume 2*. Athena Scientific, Belmont, MA, 2000.

[4] Dimitri P. Bertsekas and Robert Gallager. *Data Networks*. Prentice-Hall, Upper Saddle River, NJ, 1992.

[5] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, Belmont, MA, 1997.

[6] Dimitri P. Bertsekas and John N. Tsitsiklis. *Introduction to Probability*. Athena Scientific, Belmont, MA, 2002.

[7] Robert G. Dial. A Probabilistic Multipath Traffic Assignment Model which Obviates Path Enumeration. *Transportation Research*, 5:83–111, 1970.

[8] R.J. Duffin. Topology of Series-Parallel Networks. *Journal of Math Analysis and Applications*, 10:303 318, 1965.

[9] Salah E. Elmaghraby. *Activity Networks: Project Planning and Control by Network Models*. John Wiley and Sons, New York, NY, 1977.

[10] David Eppstein. Finding the k Shortest Paths. *SIAM Journal on Computing*, 28(2):652-673, 1998.

[11] George S. Fishman. Estimating Network Characteristics in Stochastic Activity Networks. *Management Science*, 31(5):579–593, 1985.

[12] H. Frank. Shortest Paths in Probabilistic Graphs. *Operations Research*, 17:583–599, 1969.

[13] H. Frank and S.L. Hakimi. Probabilistic Flows Through a Communication Network. *IEEE Transactions on Circuit Theory*, CT-12:413–414, 1965.

[14] A.M. Frieze and G.R. Grimmett. The Shortest Path Problem for Graphs with Random Arc Lengths. *Discrete Applied Mathematics*, 10:57–77, 1985.

[15] Liping Fu. An Adaptive Routing Algorithm for In-Vehicle Route Guidance Systems with Real-Time Information. *Transportation Research B.* 35:749 765, 2000.

[16] Song Gao. Routing Problems in Stochastic Time-Dependent Networks with Applications in Dynamic Traffic Assignment. Master's thesis, Massachusetts Institute of Technology, 2002.

[17] Mark B. Garman. More on Conditioned Sampling in the Simulation of Stochastic Networks. *Management Science*, 19(1):90–95, September 1972.

[18] H.O. Hartley and A.W. Wortham. A Statistical Theory for PERT Critical Path Analysis. *Management Science*, 12(10):B469–B481, 1966.

[19] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Verlag, New York, NY, 2001.

[20] John M. Burt Jr. and Mark B. Garman. Conditional Monte Carlo: A Simulation Technique for Stochastic Network Analysis. *Management Science*, 18(3):207–217, November 1971.

[21] Peter Kall and Stein W. Wallace. *Stochastic Programming.* John Wiley and Sons, New York, NY, 1st edition, 1994.

[22] James Seong-Cheol Kang. Algorithms for Routing Problems in Stochastic Time-Dependent Networks. Master's thesis, Massachusetts Institute of Technology, 2002.

[23] V.G. Kulkarni. Shortest Paths in Networks with Exponentially Distributed Arc Lengths. *Networks*, 16:255–274, 1986.

[24] Richard Larson and Amedeo Odoni. *Urban Operations Research.* Prentice-Hall, Upper Saddle River, NJ, 1981.

[25] D.G. Malcolm, J.H. Roseboom, C.E. Clark, and W. Fazar. Applications of a Technique for Research and Development Program Evaluation. *Operations Research*, 7(5):646–669, 1959.

[26] J.J. Martin. Distribution of the Time through a Directed, Acyclic Network. *Operations Research*, 13(1):46–66, 1965.

[27] E. Miller-Hooks and H. Mahmassani. Least Possible Time Paths in Stochastic, Time-Varying Networks. *Computers and Operations Research*, 25:1107–1125, 1998.

[28] E. Miller-Hooks and H. Mahmassani. Optimal Routing of Hazardous Materials in Stochastic, Time-Varying Transportation Networks. *Transportation Research Record*, 1645:143–151. 1998.

[29] Alan V. Oppenhiem, Alan S. Willsky, and S. Hamid Nawab. *Signals and Systems*. Prentice-Hall Signal Processing Series. Prentice-Hall, Upper Saddle River, New Jersey, 2nd edition, 1997.

[30] James B. Orlin, Thomas L. Magnanti, and Ravindra K. Ahuja. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Upper Saddle River, New Jersey, 1993.

[31] Athanasios Papoulis and S. Unnikrishna Pillai. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, New York, NY, 4th edition. 2002.

[32] George H. Polychronopoulos and John N. Tsitsiklis. Stochastic Shortest Path Problems with Recourse. *Networks*. 27:133–143. 1996.

[33] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C (The Art of Scientific Computing)*. Cambridge University Press, New York, NY, 1992.

[34] J. Scott Provan. The Complexity of Reliability Computations in Planar and Acyclic Graphs. *SIAM Journal on Computing*, 15(3):694–702, August 1986.

[35] J. Scott Provan and Michael O. Ball. The Complexity of Counting Cuts and of Computing the Probability that a Graph is Connected. *SIAM Journal on Computing*, 12(4):777–788, November 1983.

[36] Pierre Robillard and Michel Trahan. The Completion Time of PERT Networks. *Operations Research*, 25:15–29. 1977.

[37] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Upper Saddle River, New Jersey, 2nd edition. 2002.

[38] Suvrajeet Sen, Rekha Pillai, Shirish Joshi, and Ajay K. Rathi. A Mean-Variance Model for Route Guidance in Advanced Traveler Information Systems. *Transportation Science*, 35(1):37–49, February 2001.

[39] C. Elliott Sigal, A. Alan B. Pritsker, and James J. Solberg. The Stochastic Shortest Route Problem. *Operations Research*, 28(5):1122–1129, 1980.

[40] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. CRC Press, Boca Raton, FL, 1986.

[41] Timothy Law Synder and J. Michael Steele. *Handbooks in Operations Research and Management Science, Volume 7*, chapter Probabilistic Networks and Network Algorithms, pages 401–424. Elsevier Science, New York, NY, 1995.

[42] Andrew S. Tanenbaum. *Computer Networks*. Prentice-Hall, Upper Saddle River, NJ, 1981.

[43] Jacobo Valdes. Parsing Flowcharts and Series-Parallel Graphs. Technical Report STAN-CS-78-682, Stanford University Computer Science Department, 1978.

[44] Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The Recognition of Series Parallel Digraphs. In *Proceedings of the Eleventh Annual ACM Symposium on the Theory of Computing*, pages 1–12. ACM, 1979.

[45] Leslie G. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[46] Eric W. Weisstein. Fourier Transform. World of Mathematics (http://mathworld.wolfram.com/FourierTransform.html).

[47] Jin Y. Yen. Finding the k Shortest Loopless Paths in a Network. *Management Science*, 17(11):712–716, July 1971.