THE ROBUST SHORTEST PATH PROBLEM WITH INTERVAL DATA UNCERTAINTIES

A THESIS

SUBMITTED TO THE DEPARTMENT OF INDUSTRIAL ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCES

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Abdullah Sıddık Karaman

July, 2001

ABSTRACT

THE ROBUST SHORTEST PATH PROBLEM WITH INTERVAL DATA UNCERTAINTIES

Abdullah Sıddık Karaman
M.S. in Industrial Engineering
Supervisor: Assoc. Prof. Mustafa Ç. Pınar
July, 2001

In this study, we investigate the well-known shortest path problem on directed acyclic graphs under arc length uncertainties. We structure data uncertainty by taking the arc lengths as interval ranges. In order to handle uncertainty in the decision making process, we believe that a robustness approach is appropriate to use. The robustness criteria we used are the minimax (absolute robustness) criterion and the minimax regret (relative robustness) criterion. Under these criteria, we define and identify paths which perform satisfactorily under any likely input data and give mixed integer programming formulation to find them. In order to simplify decision making, we classify arcs based on the realization of the input data. We show that knowing which arcs are always on shortest paths and which arcs are never on shortest paths we can preprocess a graph for robust path problems. Computational results support our claim that the preprocessing of graphs helps us significantly in solving the robust path problems.

Key words: Shortest Path Problem, Directed Acyclic Graphs, Layered Graphs, Interval Data, Robust Optimization

ÖZET

ARALIK SAYILAR BELİRSİZLİĞİNDE EN KISA YOL PROBLEMİ

Abdullah Sıddık Karaman
Endüstri Mühendisliği Bölümü Yüksek Lisans
Tez Yöneticisi: Doç. Mustafa Ç. Pınar
Temmuz, 2001

Bu çalışmada, verileri aralık sayılarla ifade edilen yönlü çevrimsiz çizgelerde en kısa yol problemi incelenmiştir. Veriler belirsiz olduğu için amaç dayanıklı çözümler üretmektir. Dayanıklılık ölçütü olarak enfazlayı enazlama ve enfazla kaybı enazlama kullanılmıştır. Bu kriterler kullanılarak her veriye göre iyi sonuç veren yollar tanımlanmış ve bunları bulan karışık tamsayı programlama formülasyonları verilmiştir. Arklar, verilere bağlı olarak, hangilerinin en kısa yol üzerinde olup olamayacağına göre sınıflandırılmıştır ve bu sınıflandırmanın dayanıklı yol problemleri için bir ön işlem olduğu gösterilmiştir. Hesaplama sonuçları bu ön işlemin dayanıklı yol problemlerinin çözümü kolaylaştırdığı tezimizi destekler.

Anahtar sözcükler: En Kısa Yol Problemi, Yönlü Döngüsüz Çizgeler, Katmanlı Çizgeler, Aralık Sayılar, Dayanıklı Eniyileme

to my family

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Assoc. Prof. Mustafa Ç. Pınar for his supervision, suggestions, trust and understanding to bring this thesis come to an end.

I am grateful to Asst. Prof. Oya Ekin Karaşan for her interest, suggestions and help. I also would like to thank her for reading and reviewing this thesis.

I am indebted to Asst. Prof. Ezhan Karaşan for accepting to read and review this thesis and for his suggestions.

I would like to take this opportunity to thank Burhaneddin Sandıkçı for being such a good friend in all my school life. I would also like to thank my officemate Rabia Kayan for her friendship, help and moral support. I extend my sincere thanks to M. Çağrı Gürbüz, Mümin Kurtuluş, Ayten Türkcan, Banu Yüksel, Savaş Çevik, Onur Boyabatlı and Arz Pekmezci for their friendship, help and encouragement.

My special thanks go to my dearest parents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis, we investigate the well-known shortest path problem on directed acyclic graphs where the input data of the problem are uncertain. What is meant by uncertainty is that there is a range of possible realizations for each data, but the actual realizations are not known. We express this range of realizations as an interval range. The deterministic version of the problem can be solved in polynomial time. However, if there is significant data uncertainty, the deterministic approach can be far from sufficient. New and appropriate criteria and models are needed in order to handle uncertainty.

The traditional motivation for studying the shortest path problem on directed acyclic graphs with interval data comes from helping a motorist who sets out to drive from some location in a city to another. The aim is to determine a path that minimizes the travel time (distance, cost) where the traffic conditions at various roads are uncertain because of the presence of accidents, traffic jams at peak hours and construction projects etc. Since the driver does not have full information of the roads, she/he considers only a subset of roads in deciding her/his route choice in a robust manner [8].

Mathematical programming models usually have the problem of imprecise data in building a real-world system. Cost of resources, demand for the products, returns of financial instruments are examples of data that are uncertain. We encounter different ways of dealing with uncertainty in the literature. One of the ways is the "sensitivity analysis" which is a post-optimality tool. The goal of this study is to discover the impact of data perturbations on a model. It just measures the sensitivity of a solution to changes in the input data. Another way of dealing with uncertainty is based on the pro-active approaches. This approach can be classified according to the environment it is used in. There are two different kinds of environments: "Risk" and "Uncertainty". In risk situations, the link between the decisions and outcomes are probabilistic. Stochastic optimization is used to optimize the expected value of a single objective. However, in uncertainty situations it is impossible to attribute probabilities to the possible outcomes of any decision. A thorough study of literature with uncertain data can be found in [12].

There are many different ways of dealing with uncertainty in data. One way is converting the problem into risk and using probability tools. In this case, knowledge of probability distribution functions are required and probabilities are difficult to estimate. Also, if probabilities are assigned successfully, unless independence and no correlation assumed, it becomes computationally very difficult to solve the problem. It is also possible to transform the problem into a certainty problem by the use of subjective estimation of most likely numbers. However, a solution which is optimal with respect to these values yields a quite poor performance when evaluated relative to the actual realized data. Yu said: "Ample evidence exist in research literature that for decision making environments in the presence of significant data uncertainty in the input data of the decision model, either the deterministic optimization or the stochastic optimization approach may not accurately represent the aim of decision maker (see Gupta and Rosenhead [6], Rosenhead et al. [9], Sengupta [10], Kouvelis et al. [7]and Daniels and Kouvelis [3]) [14]".

Kouvelis and Yu [8] motivate the use of robustness approach to decision making in environments of significant data uncertainty. The aim of this approach is to find decisions that will have a reasonable objective value under any likely input data. They have demonstrated the applicability of this framework on several combinatorial optimization problems and dealt with the characterization of algorithmic complexity of these problems. A comprehensive treatment of the state of art in robust discrete optimization and extensive references can be found in this book.

In the present thesis, in order to handle uncertainty in the arc lengths, we applied the robust optimization framework to our problem. We structure data uncertainty by taking the arc lengths as intervals defined by known lower and upper bounds and do not assume any probability distribution. This way of defining arc lengths is easy to model when compared to stochastic methods which requires knowledge of probability distribution functions. The robustness criteria we used is the minimax (absolute robustness) criterion and minimax regret (relative robustness) criterion. We refer to the problems as "absolute robust shortest path problem" and "relative robust shortest path problem". The absolute robust shortest path problem is defined as finding among all paths the one that minimizes the maximum path length from origin to destination over all realizable input data. The relative robust shortest path problem is defined as finding among all paths the one that, over all realizable input data, minimizes the maximum deviation of the path length from the optimal path length of the corresponding realization. In the first one, the problem yields very conservative solutions based on the anticipation that the worst-case will happen. In the latter one, the decision is less conservative, since it allows benchmarking of the performance of the decision against the the best possible outcome under any realization of arc lengths.

Kouvelis and Yu [8] have studied the robust shortest path problems under arc length uncertainties. Different from our problem, they structure the uncertainty by a discrete scenario set, where each scenario represents a potential realization of the arc lengths. They prove that the robust shortest path problems are NP-complete for a bounded number of scenarios and becomes strongly NP-hard for an unbounded number of scenarios. Moreover, they conjecture that the robust path problems with interval data are also NP-complete. In solving these problems, they suggest a branch-and-bound procedure with both upper and lower bounds generated by a surrogate relaxation. They have shown that this is an effective method in practice.

A similar work to ours was done by Yaman [13]. She has studied the longest path problem on directed acyclic graphs with interval data. She defined new optimality concepts in finding a longest path in a graph based on the realizations of arc lengths. A characterization of these optimal solutions and polynomial time algorithms to find them in special cases can be found in this study. Further, she derived the mixed integer programming formulation of the relative robust longest path problem with interval data. However, Yaman did not conduct computational experiments on these problems.

Averbakh [2] presented the first example of a combinatorial optimization problem that is NP-hard in the scenario-represented uncertainty but is polynomially solvable in the case of interval representation of uncertainty. He studied robust version of the problem of selecting p elements of minimum total weight out of a set of m elements where the weights of the elements are represented as interval ranges. He has proved that the problem is NP-hard in the case of arbitrary finite set of possible scenarios, even with only two scenarios but polynomially solvable in the case of interval representation of uncertainty.

In this study, we continue the investigation initiated in Yaman [13]. We define the robust path problems with interval data and give mixed integer programming formulations of these problems. Then, we would like to distinguish paths that are shortest for all realizations and paths that are shortest for some realizations. Based on this analysis of paths, we derive some basic results for robust path problems. Since the number of paths in the graph grows exponentially with the number of nodes in the graph, this does not have a practical use when the number of nodes in a graph is very large. Therefore, we make a similar analysis of arcs which can be done in polynomial time. We show that, knowing which arcs are on shortest paths for all realization of data and which arcs are on shortest paths for some realization of data, we can preprocess a given graph for robust path problems. In other words, we can eliminate arcs from the problem that can not be on robust paths. By doing so, we solve the robust path problems on a restricted feasible set of the problem. Then, we show in practice that, this reduction of the feasible set helps us significantly in solving the robust path problem.

The rest of the thesis is organized as follows: In chapter 2, we give the formal definitions of absolute and relative shortest path problems in directed acyclic graphs with interval data. We derive the mixed integer programming formulation to find the relative robust path in a graph. Then, we make an analysis of paths and derive some basic results of robust path problems. Further, we extend our investigation to arcs based on the realization of data. In chapter 3, we present our computational results. Finally, we give conclusions in chapter 4.

Chapter 2

Shortest Path Problem with Interval Data

In this chapter, we consider the robust version of shortest path problem on directed acyclic graphs under arc length uncertainties. There are n nodes in the graph where 1 is the origin node and n is the destination node. The deterministic version of the problem can be stated as follows: Given a graph $G = (V, A)$ with node set V, and arc set A, a nonnegative length c_a associated with each arc $a \in A$, the origin node 1 and the destination node n, the shortest path problem is to find a path of minimum total length from 1 to n. The problem can be formulated as follows:

$$\begin{aligned}
&\min \sum_{(i,j)\in A} l_{ij} y_{ij} \\
&\text{subject to} \\
&\quad -\sum_{i\in\Gamma^-(j)} y_{ij} + \sum_{k\in\Gamma^+(j)} y_{jk} = b_j \quad j = 1, 2, .., n \\
&\quad y_{ij} \in \{0, 1\} \quad \forall (i, j) \in A
\end{aligned}$$

where l_{ij} represents the length of arc (i, j) and y_{ij}, b_j, $\Gamma^-(j)$, $\Gamma^+(j)$ are defined as

$$y_{ij} = \begin{cases} 1 & \text{if arc } (i, j) \text{ is on the path} \\ 0 & \text{otherwise} \end{cases}$$

$$b_j = \begin{cases} 1 & \text{for } j = 1 \\ 0 & \text{for } j \neq 1, n \\ -1 & \text{for } j = n \end{cases}$$

$\Gamma^-(j) = \{i \in V : (i,j) \in A\}$, and $\Gamma^+(j) = \{k \in V : (j,k) \in A\}$. In this formulation, the first constraint set represents the network flow constraints and the second constraint set forces the variables to take on binary values. A vector y satisfying the above set of constraints defines a path in the graph. The problem is to find a path of minimum total length between the origin and destination nodes. In this problem, the integer requirements can be relaxed due to the unimodularity property of the constraint matrix. All the arc lengths are assumed to be known in advance. This problem is one of the simplest and well-studied combinatorial optimization problems, and it is a special case of the class of network flow problems with a single source and a single sink. An efficient $O(|V|^2)$ time labeling algorithm in general networks was given by Dijkstra [5]. Other polynomial time algorithms with complexity $O(m)$ time where m is the number of arcs can be found in [1].

Here, what is meant by arc length uncertainty is that there is a range of possible realizations of arc lengths. This range may be based on pessimistic and optimistic estimates of arc lengths or on likely deviations from average values. In order to characterize it, we take the arc length values as interval ranges. To be more precise, arc (i,j) has length l_{ij} within a given lower bound $\underline{l}_{ij}$ and an upper bound $\overline{l}_{ij}$ i.e., $\underline{l}_{ij} \leq l_{ij} \leq \overline{l}_{ij}$. Each value in the interval can be realized by some positive probability but no probability distribution is assumed for the arc lengths. l_{ij} takes an arbitrary value in the interval $[\underline{l}_{ij}, \overline{l}_{ij}]$. A realization of all arc lengths is called a scenario s. l_{ij}^s denotes the length of arc (i,j) in scenario s.

Let P be the set of all paths from 1 to n. We denote by l_p^s the length of path p in scenario s, and $\overline{l}_p$, $\underline{l}_p$ denote the length of path p when the lengths of all arcs on path p are at upper bounds and the lengths of all arcs on path p are at lower bounds, respectively.

In decision making environments where there is a significant data uncertainty neither the deterministic approach nor the stochastic optimization can accurately represent the aim of the decision maker. In this type of environment, we believe that a "robustness approach" is more appropriate to use. This approach assumes inadequate knowledge of the decision maker about the random state of nature and develops a decision that hedges against the worst case that may arise. It does not ignore uncertainty, even it takes a pro-active step in response to the fact that predicted values of the uncertain parameters will not occur. A robust solution is defined to be the one which performs rather well whatever data is realized. Under any likely input data, the aim is to find a solution that will have a reasonable objective value. Among the many possible robustness criteria, we choose the "minimax" and "minimax regret" criteria to apply to our problem. Minimax (absolute robustness) criterion finds a decision for which the maximum objective value of the solution taken across all possible input data is as low as possible. This criterion gives a solution based on the prediction that the worst case will happen. Another criterion we used is the minimax regret (relative robustness) criterion. Regret can be defined as the difference between the cost of a specific decision and the corresponding cost of the optimal decision for a specific realization. Then, minimax regret can be explained as to choose the decision with the least maximum regret. These robustness criteria are introduced in [8]. In this book, the authors have motivated the robustness approach to decision making of significant data uncertainty in contrast with deterministic and stochastic optimization. They have listed several drawbacks of deterministic and stochastic approach in the face of data uncertainty.

The rest of the chapter is organized as follows: In section 2.1, we give the formal definition of absolute robust path problem. In section 2.2, we define what we mean by a relative robust path and derive the mixed integer programming formulation of the problem. Then, in section 2.3 we make a short analysis of paths and distinguish which are shortest for all realizations and which are shortest for some realization. Then, we derive some basic results about robust path problems. Finally, in section 2.4, we distinguish arcs which are always on shortest paths, and those which are never on shortest paths.

Then, we show that knowing which arcs are never on shortest paths, we can preprocess a given graph for robust path problems.

2.1 Absolute Robustness

Next, we would like to define what we mean by a robust path. A robust path is defined to be the one which performs satisfactorily whatever data is realized. In this section, we select the minimax (absolute robust) criterion to find a robust path. This criterion will select a path for which the maximum path length taken across all possible realizations is as low as possible. In other words, we would like to find a path that minimizes the maximum path length between the origin and destination nodes. The mathematical formulation of the problem is:

$$\begin{aligned}
&\min_{y} \max_{s \in S} \sum_{(i,j)\in A} l_{ij}^{s} y_{ij} \\
&\text{subject to} \\
&\quad -\sum_{i\in\Gamma^{-}(j)} y_{ij} + \sum_{k\in\Gamma^{+}(j)} y_{jk} = b_j \quad j = 1, 2, .., n \\
&\quad y_{ij} \in \{0,1\} \quad \forall (i,j) \in A
\end{aligned}$$

where S denotes the set of all possible scenarios.

Kouvelis and Yu have studied the absolute robust path problem. In their study, they used a scenario planning approach to characterize uncertainty. A specific input data represents a possible realization. They have proved that the absolute robust path problem is NP-complete even in layered networks of width 2 and with only 2 scenarios. Further, they have showed that the problem can be solved in pseudo-polynomial time for layered networks with bounded scenario set and the problem is strongly NP-hard for an unbounded number of scenarios.

In our case, in order to find a solution to the absolute robust path problem, it is enough to consider the unique scenario where the lengths of all arcs on the graph are set to their upper bounds since the maximum path length corresponds to this unique scenario. Then we can find the absolute robust path by finding the shortest path in the graph under this scenario. So, the problem reduces to:

$$
\begin{aligned}
&\min \sum_{(i,j)\in A} \bar{l}_{ij} y_{ij} \\
&\text{subject to} \\
&\quad -\sum_{i\in\Gamma^-(j)} y_{ij} + \sum_{k\in\Gamma^+(j)} y_{jk} = b_j \quad j = 1, 2, .., n \\
&\quad y_{ij} \in \{0,1\} \quad \forall (i,j) \in A
\end{aligned}
$$

This is identical to the conventional shortest path problem. Then, the absolute robust path problem can be solved in polynomial time by algorithms given in aforementioned studies.

Under the absolute robustness criterion, the solutions are not sensible to the realization of data. Use of this approach yields very conservative solutions based on the anticipation that the worst case might well happen. Under this criterion, the main concern is how to hedge against the worst possible contingency.

2.2 Relative Robustness

In the previous section, the absolute robust path problem yields a very conservative solution based on the prediction that the worst case will happen. However, in reality a solution with a reasonable objective value under any likely input data will be satisfactory for a decision maker. In this section, we would like to find a path that the maximum difference between the length of this path and length of the shortest path for the corresponding realization of input data is smallest. Saying differently, a solution that exhibits the smallest worst case deviation from optimality over all potential realizations. This solution allows the benchmarking of the performance of the decisions against the best possible outcome under any data set. Next, we give a formal definition of what we mean by robust deviation and then derive a mixed integer programming formulation of relative robust path problem.

Definition 2.1 *The* **robust deviation** *for a path p is defined as the difference between the length of path p and the length of the shortest path in the graph for a specific realization of arc lengths, i.e., $d_p = l_p - l_{p^*}$ where d_p denotes the robust deviation and p^* denotes the shortest path in the graph.*

Definition 2.2 *A path p is said to be a* **relative robust path** *if it has the least maximum robust deviation among all paths. i.e., relative robust path $p^r = \arg\min_{p\in P} \max_{s\in S} l_p^s - l_{p^*(s)}^s$ where $p^*(s)$ denotes the shortest path in scenario s.*

Kouvelis and Yu [8] have also studied the relative robust path problem under arc length uncertainties. They adopt a scenario planning approach to characterize uncertainty. They have shown that the relative robust path problem is NP-complete even in layered networks of width 2 and with only 2 scenarios. Also, they have proved that the problem is strongly NP-hard for an unbounded number of scenarios.

Define y_{ij}'s as follows:

$$y_{ij} = \begin{cases} 1 & \text{if arc } (i,j) \text{ is on the path} \\ 0 & \text{otherwise} \end{cases}$$

The mathematical formulation of the problem is:

$$\begin{aligned} &\min_{y} \max_{s\in S} \Big(\sum_{(i,j)\in A} l_{ij}^s y_{ij} - x^s \Big) \\ &\text{subject to} \\ &\quad -\sum_{i\in\Gamma^-(j)} y_{ij} + \sum_{k\in\Gamma^+(j)} y_{jk} = b_j \quad j = 1,2,..,n \\ &\quad y_{ij} \in \{0,1\} \quad \forall (i,j) \in A \end{aligned}$$

where S denotes the set of all possible scenarios and x^s is the length of shortest path in the graph under scenario s.

In our case, in order to find the relative robust path in a graph, we need to only consider the scenario which makes the robust deviation maximum. Since robust deviation is defined as the difference between the length of path p and the length of the shortest path in the graph, this scenario corresponds to a path p in which the lengths of all arcs on p are at upper bounds and the lengths of all other arcs at their lower bounds. This implies that we need to consider only a finite number of scenarios which is equal to the number of paths in the graph. However, the number of paths in a graph grows exponentially with the number of nodes in the graph.

Kouvelis and Yu [8] have conjectured that the relative robust path problem with interval data is also NP-complete.

Next, we present a mixed integer programming formulation to find the relative robust path in a graph. In the formulation, the length of arc (i, j) is defined as $l_{ij} = \underline{l}_{ij} + (\overline{l}_{ij} - \underline{l}_{ij})y_{ij}$ for a given vector y. This is because when $y_{ij} = 1$ the length of arc (i, j) is at its upper bound on path p defined by y. All the lengths of other arcs with $y_{ij} = 0$ are at their lower bounds.

Let x_j be the shortest distance from node 1 to node j. We have the following set of constraints which specifies shortest distances from node i to node j based on whether arc (i, j) is on the path or not:

$$x_j \leq x_i + \underline{l}_{ij} + (\overline{l}_{ij} - \underline{l}_{ij})y_{ij} \quad \forall (i, j) \in A$$

So, x_n is the length of the shortest path in the graph under the scenario defined by y. The objective is to find a path p for which the difference between the length of path p and the length of shortest path in the graph is the smallest when the lengths of all arcs on path p are at their upper bound and the lengths of all other arcs are at their lower bounds.

The mixed integer programming formulation of the relative robust path is as follows:

(RRP)

$$
\begin{aligned}
&\min \sum_{(i,j)\in A} \bar{l}_{ij} y_{ij} - x_n \\
&\text{subject to} \\
&\quad x_j \leq x_i + \underline{l}_{ij} + (\bar{l}_{ij} - \underline{l}_{ij}) y_{ij} \quad \forall (i,j) \in A \\
&\quad - \sum_{i\in\Gamma^-(j)} y_{ij} + \sum_{k\in\Gamma^+(j)} y_{jk} = b_j \quad j = 1,2,..,n \\
&\quad x_1 = 0 \\
&\quad y_{ij} \in \{0,1\} \quad \forall (i,j) \in A \\
&\quad x_j \geq 0 \quad j = 1,2,..,n
\end{aligned}
$$

The second constraint in the formulation ensures that the resulting y vector defines a path in the graph and the third constraint prevents an unbounded solution.

Solutions under this criterion will be less conservative when compared with absolute robustness. The deviation from optimality is a measure that allows the benchmarking of the decision against the best possible outcome. The robust decision is the one which keeps its performance close to best under any scenario. It is relatively insensitive to the potential realizations of parameters.

2.3 Paths

In the previous section, we have seen that the relative robust path problem is much harder than the conventional shortest path problem. In solving the relative robust path problem, reducing the solution space becomes an important issue. In this section, we would like to make an analysis of paths according to the realizations of arc lengths. We classify paths as if they are shortest for all realizations of arc lengths (permanent paths), if they are shortest for some realizations of arc lengths (weak paths) and if they are never shortest paths. These concepts are defined by Demir *et al.* [4]. Then knowing which paths are never shortest, we can preprocess a given graph for relative robust path problem. In other words, we can look for a relative robust path only among candidate paths. Therefore, our search space will be now smaller than the original search space of the relative robust path problem.

2.3.1 Permanent Paths

In this section, we would like to find a path which is shortest for all realizations of arc lengths. We call such a path a *permanent path* and give a characterization of it.

Definition 2.3 *A path is said to be a* **permanent path** *if it is a shortest path for all realizations of arc lengths.*

A necessary and sufficient condition for a path to be permanent is:

Theorem 2.1 *A path is a permanent path if and only if it is one of the shortest paths when the lengths of all arcs on this path are at their upper bounds and the lengths of all remaining arcs are at their lower bounds.*

Proof
By definition, if a path is a permanent path then it is a shortest path for all realizations of arc lengths.

If a path p is a shortest path when the lengths of all arcs on p are at their upper bounds and the lengths of all the remaining arcs are at their lower bounds, we have the following inequality for a path $p^{'} \in P$:

$$\sum_{(i,j)\in p\setminus p'} \overline{l}_{ij} + \sum_{(i,j)\in p\cap p'} \overline{l}_{ij} \leq \sum_{(i,j)\in p'\setminus p} \underline{l}_{ij} + \sum_{(i,j)\in p\cap p'} \overline{l}_{ij}.$$

Since $\sum_{(i,j)\in p\setminus p'} l_{ij} \leq \sum_{(i,j)\in p\setminus p'} \overline{l}_{ij}$ and $\sum_{(i,j)\in p'\setminus p} l_{ij} \geq \sum_{(i,j)\in p'\setminus p} \underline{l}_{ij}$ we have the following inequality:

$$\sum_{(i,j)\in p\setminus p'} l_{ij} + \sum_{(i,j)\in p\cap p'} \overline{l}_{ij} \leq \sum_{(i,j)\in p'\setminus p} l_{ij} + \sum_{(i,j)\in p\cap p'} \overline{l}_{ij}.$$

If we take arbitrary values of arcs (i,j) in $p \cap p'$, we have

$$\sum_{(i,j)\in p\setminus p'} l_{ij} + \sum_{(i,j)\in p\cap p'} l_{ij} \leq \sum_{(i,j)\in p'\setminus p} l_{ij} + \sum_{(i,j)\in p\cap p'} l_{ij}.$$

So, $l_p \leq l_{p'}$ for all realization of arc lengths. Hence, p is a permanent path. □

We can check whether a given path p is permanent or not by simply setting the lengths of all arcs on p to their upper bounds and setting the lengths of all the remaining arcs to their lower bounds, then find a shortest path in the graph. If the shortest path in the graph has the same length as p, then p is a permanent path. Otherwise, p can not be a permanent path.

Permanent solutions remove uncertainty from the decision making process. If such a solution is found, then we can decide on this solution without any thought of suboptimality. Moreover, in our case, the permanent path is both the absolute robust path and the relative robust path. It is the absolute robust path, because it is one of the shortest paths under any realization of arc lengths. It is also the relative robust path because it is one of the shortest paths when the lengths of all arcs on this path are at their upper bounds and lengths of all other arcs are at their lower bounds. This scenario corresponds to the scenario which makes the robust deviation maximum and the deviation here is zero. So, this path is also the relative robust path.

In order for a permanent path to exist in a graph, the interval estimates should be as small as possible. It is more likely for a permanent path to exist if the upper bounds and the lower bounds are closer to each other. On the other hand, the larger the intervals of arc lengths, the more arc realizations are possible and this avoids the existence of permanent paths [11].

2.3.2 Weak Paths

We now look for a path that is shortest for some realizations of arc lengths. We call such a path a *weak path* and give a characterization of it. Then, we prove the basic result that a relative robust path is a weak path.

Definition 2.4 *A path is said to be a* **weak path** *if it is a shortest path for at least one realization of arc lengths.*

Next, we give a necessary and sufficient condition for a path to be weak.

Theorem 2.2 *A path p is a weak path if and only if it is a shortest path when the lengths of all arcs on path p are at their lower bounds and the lengths of all the remaining arcs are at their upper bounds.*

Proof

If a path p is a shortest path when the lengths of all arcs on p are their lower bounds and the lengths of all the remaining arcs are at their upper bounds, then it is a weak path by definition.

Assume a path p is a weak path. Then, it is a shortest path for at least one realization of arc lengths. Denote this realization by scenario s. Then for any $p' \in P$ we have,

$$\sum_{(i,j)\in p\backslash p'} l_{ij}^s + \sum_{(i,j)\in p\cap p'} l_{ij}^s \leq \sum_{(i,j)\in p'\backslash p} l_{ij}^s + \sum_{(i,j)\in p\cap p'} l_{ij}^s.$$

Since, $\sum_{(i,j)\in p\backslash p'} \underline{l}_{ij} \leq \sum_{(i,j)\in p\backslash p'} l_{ij}^s$ and $\sum_{(i,j)\in p'\backslash p} l_{ij}^s \leq \sum_{(i,j)\in p'\backslash p} \overline{l}_{ij}$ we have

$$\sum_{(i,j)\in p\backslash p'} \underline{l}_{ij} \leq \sum_{(i,j)\in p\backslash p'} l_{ij}^s \leq \sum_{(i,j)\in p'\backslash p} l_{ij}^s \leq \sum_{(i,j)\in p'\backslash p} \overline{l}_{ij}.$$

If we add $\sum_{(i,j)\in p\cap p'} \underline{l}_{ij}$ to both sides of the first and the last term, we get,

$$\sum_{(i,j)\in p\backslash p'} \underline{l}_{ij} + \sum_{(i,j)\in p\cap p'} \underline{l}_{ij} \leq \sum_{(i,j)\in p'\backslash p} \overline{l}_{ij} + \sum_{(i,j)\in p\cap p'} \underline{l}_{ij}.$$

So, p is a shortest path when the lengths of all arcs on p are at their lower bounds and the lengths of all the remaining arcs are at their upper bounds. $\Box$

Based on the above analysis of paths, we now derive the basic result for robust path problems. Clearly, an absolute robust path is a weak path. We now show that, a relative robust path is also a weak path.

Proposition 2.1 *A relative robust path is a weak path.*

Proof

Let p be a path, which is not weak. Let p' be another path which is a shortest path when the lengths of all arcs on p are at their lower bounds and the lengths of the remaining arcs are at their upper bounds. Then, $l_p > l_{p'}$ for all realizations of arc lengths. Consider the scenario s^* for path p' when the lengths of all arcs on p' are at their upper bounds and the lengths of all the remaining arcs are at their lower bounds. Then, we have:

$$l_{p'}^{s^*} - l_{p^*(s^*)}^{s^*} < l_p^{s^*} - l_{p^*(s^*)}^{s^*} \leq \max_{s \in S} l_p^s - l_{p^*(s)}^s.$$

So p can not have the least maximum regret. Hence, p cannot be a relative robust path. $\Box$

The size of the weak solution set depends on the gap between the lower and upper bounds. The wider the gap between bounds, the more arc realizations are possible. Therefore, we have a larger weak solution set.

Since the number of paths in the graph grows exponentially with the number of nodes in the graph, the analysis of paths does not have a practical use when the number of nodes is large. Instead, we can make a similar analysis of arcs which can be done in polynomial time.

2.4 Arcs

In this section, we make an analysis of arcs which is similar to what we make for paths. We classify arcs as if they are on shortest paths for some realizations and if they are never on shortest paths. We call an arc a *weak arc* if it is on a shortest path for some realization of arc lengths and *non-weak arc* if it is never on a shortest path. We show that we can make this analysis in polynomial time. Further, we can use this information of arcs in solving the relative robust path. If we can determine which arcs are never on shortest paths, we can eliminate the paths using these arcs from the graph, since a relative robust path is a weak path. We investigate the arc problems on two different types of graphs: complete graphs and layered graphs.

2.4.1 Arc Problems on Complete Graphs

In this section, we investigate the arc problems on complete graphs. A **complete graph** is an acyclic graph in which each pair of distinct nodes i and j for $i < j$ is joined by arc (i, j). First, we give a formal definition of what we mean by a weak arc. Then we present two polynomial time procedures for which the eliminated arcs from these procedures are non-weak arcs. We also give a mixed integer programming formulation to check whether a given arc is weak or not.

Definition 2.5 *An arc (i, j) is said to be a* **weak arc** *if it is on one of the weak paths.*

We now present two procedures which eliminate arcs that cannot be on weak paths. The procedures are based on the following necessary condition of an arc in order to be weak.

Proposition 2.2 *If arc (i, j) is weak, then it is weak in the subgraph generated by node 1 up to node j and it is weak in the subgraph generated by node i up to node n.*

Proof
Assume arc (i, j) is not weak in the subgraph generated by node 1 up to node j. Then, the path p uses arc (i, j) is not a weak path in the subgraph when the lengths of all arcs on p are at their lower bounds and the length of the remaining arcs are at their upper bounds. Then, there exists another path $p' \in P$ such that

$$\sum_{(k,l)\in p} \underline{l}_{kl} > \sum_{(k,l)\in p'} \overline{l}_{kl}.$$

Let p' be an arbitrary path which has the same arcs as p in the partial path from node j to node n, i.e., the partial path p_{j-n}. Then, we have:

$$\sum_{(k,l)\in p\backslash p_{j-n}} \underline{l}_{kl} + \underline{l}_{p_{j-n}} > \sum_{(k,l)\in p'\backslash p_{j-n}} \overline{l}_{kl} + \underline{l}_{p_{j-n}}.$$

So, p cannot be weak in the whole graph. Since p' is picked arbitrarily, arc (i,j) can not be weak. $\square$

So, based on Proposition 2.2, we can decide if arc (i,j) is non-weak by checking whether it is weak in the subgraph generated by node 1 up to node j and it is weak in the subgraph generated by node i up to node n. We can accomplish this task as follows. In order to check an arc (i,j) is weak or not in the subgraph generated by node i up to node n, we start with node i and consider each node one by one to node n. We set all the arc lengths in the subgraph to their upper bounds except the arc (i,j) and the arcs that are emanating from node j, i.e., arcs (j,k) in the set $\Gamma^+(j) = \{k \in V : (j,k) \in A\}$. We set the lengths of these arcs to their lower bounds. By doing so, our procedure will construct a weak path using arc (i,j), if one exist. In this realization of arc lengths, we find a shortest path from node i to node $j+1$. We favor the path that uses arc (i,j), if there exist two equal length shortest paths. We have two possibilities: If the shortest path between nodes i and $j+1$ uses arc (i,j), we set the lengths of arcs that are emanating from node $j+1$ to their lower bounds, i.e., arcs in the set $\Gamma^+(j+1) = \{k \in V : (j+1,k) \in A\}$ and continue our investigation with node $j+2$. On the other hand, if the shortest path does not use arc (i,j), we do not change any of the arc lengths and continue our investigation with node $j+2$. By going through the same steps, we continue our investigation till node n. After we reach node n, if the shortest path does not use (i,j), by proposition 2.2, we can definitely say that arc (i,j) can not be weak, since if there is not a weak path in the subgraph that uses arc (i,j), then there is not a weak path in the whole graph that uses arc (i,j). On the other hand, if the shortest path uses arc (i,j), we cannot conclude anything since this proposition is only a necessary condition.

Determining whether an arc (i,j) is weak in the subgraph generated by node 1 up to node j is equivalent to the above procedure applied to the subgraph generated by node 1 up to node j.

Next, we present the first procedure in order to check whether an arc (i,j) is weak or not.

Procedure Forward

1. Generate the subgraph starting from node i to node n.
2. Set $l_{ij} = \underline{l}_{ij}$, $l_{jk} = \underline{l}_{jk}$ $\forall k \in V$ and $l_{kl} = \overline{l}_{kl}$ $\forall$ other (k,l) $\in$ A.
3. For node $j+1$ to node n
    (a) Find a shortest path between nodes i and $j+1$.
        i. If the shortest path uses arc (i,j), then set $l_{j+1,k} = \underline{l}_{j+1,k}$ $\forall k \in V$.
4. If the shortest path between nodes i and n does not use (i,j), then arc (i,j) is not a weak arc.

We have a proposition which states that the procedure Forward distinguishes non-weak arcs.

Proposition 2.3 *The arcs eliminated by the procedure Forward are non-weak arcs.*

Proof
Simply follows from proposition 2.2. □

Now, we present the second procedure which is similar to the first one.

Procedure Backward

1. Generate the subgraph starting from node 1 to node j.
2. Set $l_{ij} = \underline{l}_{ij}$, $l_{ri} = \underline{l}_{ri}$ $\forall r \in V$ and $l_{kl} = \overline{l}_{kl}$ $\forall$ other (k,l) $\in$ A.

3. For node $i-1$ to node 1
   (a) Find a shortest path between nodes $i-1$ and j.
      i. If the shortest path uses arc (i,j), then set $l_{r,i-1} = \underline{l}_{r,i-1}$ $\forall r \in V$.
4. If the shortest path between nodes j and 1 does not use (i,j), then arc (i,j) is not a weak arc.

We have a similar proposition which states that the procedure Backward distinguishes non-weak arcs.

Proposition 2.4 *The arcs eliminated by the procedure Backward are non-weak arcs.*

The following proposition states the complexity of both procedures.

Proposition 2.5 *The running times of the procedures Forward and Backward are $O(m^2)$.*

Proof
In the worst case, we solve a shortest path problem for a given arc and it takes $O(m)$ time to find the shortest path in the graph. There are totally m arcs in the graph. $\square$

We may not determine all the non-weak arcs even if we try both of the procedures. Also, the procedures do not determine the same non-weak arcs. However, in practice, we can figure out almost all of the non-weak arcs by these procedures. Still, we have a mixed integer programming formulation to determine whether a given arc is weak or not. The formulation depends on the following characterization of weak arcs. A characterization of weak arcs is as follows.

Lemma 2.1 *An arc (i,j) is weak if and only if $\min_{p \in P_{(i,j)}} \{\underline{l}_p - l^{s_p}_{p^*(s_p)}\} = 0$, where $P_{(i,j)}$ is the set of paths using arc (i,j), s_p is the scenario in which the* *lengths of all arcs on path p are at their lower bounds and the lengths of the remaining arcs at their upper bounds.*

We now present a mixed integer programming formulation which determines whether a given arc (i,j) is weak or not.

(WA)

$$\min \sum_{(k,l)\in A} \underline{l}_{kl} y_{kl} - x_n$$

subject to

$$x_l \le x_k + \overline{l}_{kl} - (\overline{l}_{kl} - \underline{l}_{kl}) y_{kl} \quad \forall (k,l) \in A$$

$$- \sum_{k \in \Gamma^-(l)} y_{kl} + \sum_{h \in \Gamma^+(l)} y_{lh} = b_l \quad l = 1,2,..,n$$

$$y_{ij} = 1$$

$$x_1 = 0$$

$$y_{kl} \in \{0,1\} \quad \forall (k,l) \in A$$

$$x_k \ge 0 \quad k = 1,2,..,n$$

In the formulation, a vector $\mathbf{y}$ satisfying the network flow constraints and $y_{ij} = 1$ defines a path in the graph using arc (i,j). The length of arc (i,j) is defined as $l_{ij} = \overline{l}_{ij} - (\overline{l}_{ij} - \underline{l}_{ij}) y_{ij}$ for a given vector y. This is because when $y_{ij} = 1$ the length of arc (i,j) is at its lower bound on path p defined by y. All the lengths of other arcs with $y_{ij} = 0$ are at their upper bounds.

Let x_j be the shortest distance from node 1 to node j. We have the following set of constraints which specifies shortest distances from node i to node j based on whether arc(i,j) is on the path or not:

$$x_j \le x_i + \overline{l}_{ij} - (\overline{l}_{ij} - \underline{l}_{ij}) y_{ij} \quad \forall (i,j) \in A$$

So, x_n is the length of the shortest path in the graph under the scenario defined by y. The objective is to find a path p using arc (i,j) for which the difference between the length of path p and the length of shortest path in the graph is the smallest when the lengths of all arcs on path p are at their lower bound and the lengths of all other arcs are at their upper bounds.

Next, we give a theorem which characterizes weak arcs using the formulation WA.

Theorem 2.3 *Arc (i,j) is weak if and only if WA has an optimal objective value of 0.*

In this formulation, we are trying to find a path p using arc (i,j) where the lengths of all arcs on this path are at their lower bounds, and the lengths of all remaining arcs are at their bounds. Under this scenario, arc (i,j) is a weak arc if and only if the difference between the length of path p using arc (i,j) and length of the shortest path in the graph is zero. Otherwise, arc (i,j) cannot be weak.

In practice, solving a mixed integer problem to distinguish all non-weak arcs will take huge amount of time. It is more appropriate to run two polynomial time procedures to distinguish almost all the non-weak arcs.

2.4.2 Arc Problems on Layered Graphs

Here, we would like to investigate the arc problems on layered graphs. A **layered graph** is defined as one that holds the following properties. The node set can be partitioned into disjoint subsets $V = \{s\} \cup V_1 \cup V_2 \cup \ldots \cup V_m \cup \{t\}$ with $V_i \cap V_j = \emptyset$, $i \neq j$. The arcs exist only from s to V_1, from V_m to t, and from V_k to V_{k+1} for $k = 1, 2, \ldots, m-1$. Let $w = \max\{|V_k| : k = 1, 2, \ldots, m\}$, w is called the width of the layered graph. This class of graphs are special cases of general graphs. However, this is not a restricted class of graphs since every acyclic graph can be turned into a layered graph by adding dummy nodes and arcs. Figure 2.1 shows an example of an m layered graph with width 2.

In this section, we classify the arcs on layered graphs into two groups: Arcs incident at nodes s and t and intermediate arcs. First, we present a procedure to check whether an arc incident at nodes s or t is weak or not. Then, we modify that procedure to determine if an intermediate arc is weak or not. Both of the

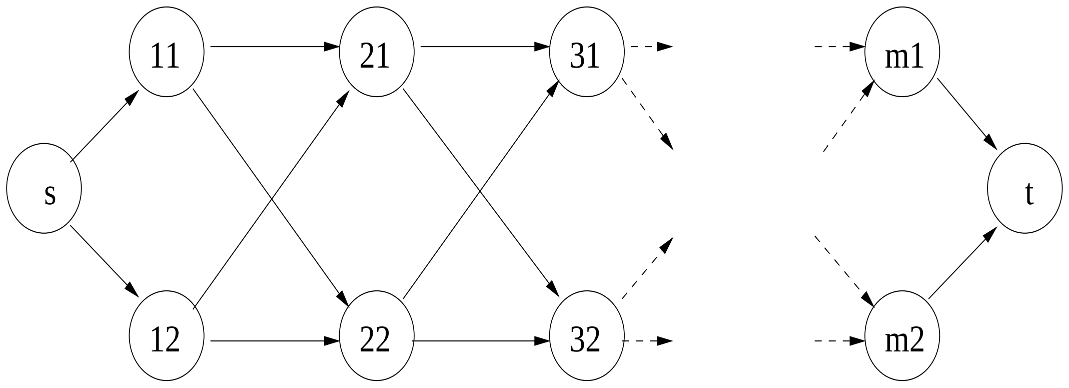

Figure 2.1: An m layered graph with width 2

procedures have polynomial running times.

Arcs incident at nodes s and t

In this section, we give a procedure which can distinguish whether a given arc which is incident at nodes s and t is weak or not. We present the procedure for arcs incident at node s. The procedure for arcs incident at node t is same as the given procedure but for the mirror version of the graph.

The procedure is based on the logic that it constructs a weak path using arcs incident at node s if one exist. The lengths of the all arcs on this path will be at their lower bounds and the lengths of all other remaining arcs are at their upper bounds. For simplicity, we present the algorithm for arc $(s, 11)$.

First, we generate the subgraph with node s, nodes in V_1, and nodes in V_2. The procedure starts with setting the lengths of all arcs that can possibly be on a path with $(s, 11)$ to their lower bounds and lengths of all other arcs to their upper bounds. Then, we find the shortest paths from node s to to all nodes in layer 2. If there are equal length paths, we favor the path that uses arc $(s, 11)$. There are three possibilities to consider.

If all the shortest paths from node s to all nodes in layer 2 uses arc $(s, 11)$, then we can say that arc $(s, 11)$ is weak. This is because, all the weak paths on the graph will use arc $(s, 11)$. Another possibility is that none of the shortest paths uses arc $(s, 11)$. Then, arc $(s, 11)$ can not be a weak arc since we can not construct a weak path path that uses arc $(s, 11)$. Finally, if some of the shortest paths from node s to nodes in layer 2 use arc $(s, 11)$, then arc $(s, 11)$ can be weak or not. In order to decide whether it is weak or not, we should continue our investigation further. We accomplish this task as follows. We shrink the graph between nodes s and nodes in layer 2. We set the lengths of arcs $(s, 2j)$ to the shortest path lengths from node s to node $2j$. Here, we label shortest paths that are at their lower bounds. At this point, we add the layer 3 to the subgraph. We decide the lengths of arcs between layer 2 and layer 3 as follows: If the shortest path between node s and nodes $2j$ in layer 2 uses arc $(s, 11)$, i.e., labeled, then we set the lengths of arcs $(2j, 3l)$ to their lower bounds since these arcs can possibly be on a weak path with arc $(s, 11)$. If the shortest paths from node s to other nodes in layer 2 do not use arc $(s, 11)$, we set the lengths of other arcs to their upper bounds since these arcs can not be on a weak path with arc $(s, 11)$. Then, we consider the new shrunk graph and find shortest paths from node s to all nodes in layer 3.

If all the shortest paths in the shrunk graph use arc $(s, 11)$, i.e., labeled, then arc $(s, 11)$ is a weak arc. If none of the shortest paths use arc $(s, 11)$, then arc $(s, 11)$ is not a weak arc. If some of the shortest paths uses arc $(s, 11)$, we shrink the graph between node s and nodes in layer 3, and add another layer to the subgraph. Then, we continue the investigation further as defined above. In the worst case, we can shrink the graph till layer t and decide whether arc $(s, 11)$ is weak or not.

Determining whether arcs incident at node t are weak or not is similar to the above procedure. Actually, it is same as this procedure applied in the mirror version of the graph.

Next, we present the procedure for arc $(s, 11)$.

Procedure

1. Generate the subgraph starting from node s, nodes in V_1 and nodes in V_2.
2. Set $l_{ij} = \overline{l}_{ij}$ $\forall (i,j) \in A$.
3. Set $l_{(s,11)} = \underline{l}_{(s,11)}$ and $l_{(11,2j)} = \underline{l}_{(11,2j)}$ for all $2j$ in V_2.
4. For nodes in $B = V_2, V_3, \ldots, V_m, t$
   (a) Find shortest paths between node s to nodes in B
      i. If all the shortest paths use arc $(s, 11)$, then arc $(s, 11)$ is a weak arc. Stop.
      ii. If none of the shortest paths uses arc $(s, 11)$, then arc $(s, 11)$ is not a weak arc. Stop.
      iii. For all shortest paths from s to $2j$ in V_2 that use arc $(s, 11)$, set $l_{(2j,3k)} = \underline{l}_{(2j,3k)}$ for all $k \in V_3$.
      iv. Shrink the graph from node s to $2j$ in V_2, set the lengths of arcs $(s, 2j)$ to the shortest path lengths from node s to node $2j$.
5. If the shortest path between nodes s and t uses arc $(s, 11)$, then arc $(s, 11)$ is weak. Otherwise, not.

We have a proposition which states the complexity of the above procedure.

Proposition 2.6 *The running time of the above procedure in an m layered graph with width w is $O(mw^2)$.*

Proof
For each node in layer k, we can find a shortest path in $O(w)$ time since there exist w paths to consider, and we have w nodes in layer k. In the worst case, we go through all the m layers in the graph. $\square$

Example 1

To clarify the above procedure, we want to apply it on the graph given in figure 2.2 which is a 3 layered graph with width 3. We want to apply the procedure for arc $(s, 12)$.

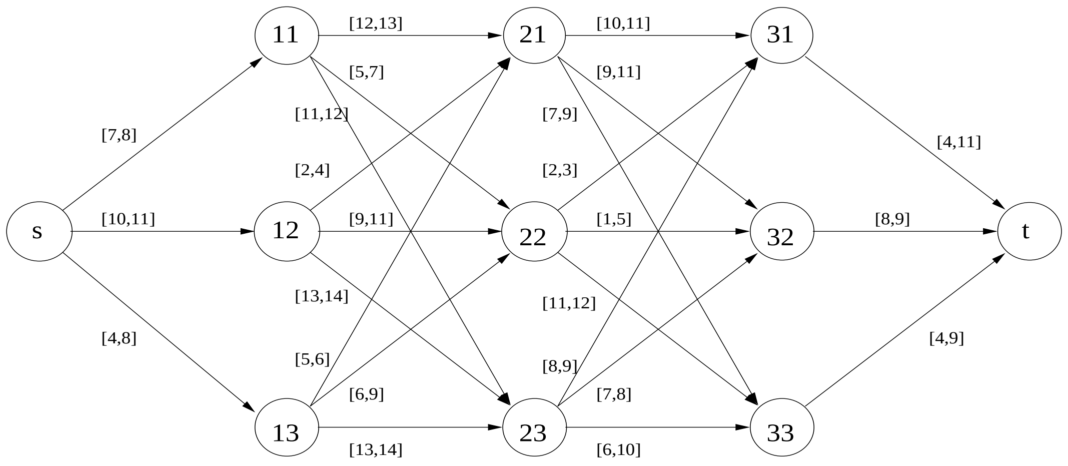

Figure 2.2: A 3 layered graph with width 3

We first generate the subgraph with nodes s, nodes in V_1 and nodes in V_2. We set the lengths of arcs $(s, 12)$, $(12, 21)$, $(12, 22)$, and $(12, 23)$ to their lower bounds and the lengths of the remaining arcs to their upper bounds. We represent the arcs at their upper bounds with dashed lines. The subgraph is given in figure 2.3. Then, we find the shortest paths from node s to all nodes in layer 2.

The shortest path from node s to node 21 uses arc $(s, 12)$, but the other shortest paths do not use this arc. So, we shrink the graph between nodes s and nodes in layer 2 and add layer 3 to the graph. We set the lengths of arcs $(21, 31)$, $(21, 32)$, and $(21, 33)$ to their lower bounds and remaining arcs to their upper bounds. The resulting graph is in figure 2.4.

We find the shortest paths from node s to all nodes in layer 3. Only, the shortest path from node s to node 33 uses arc $(s, 12)$. Then, we shrink the graph again and add node t to the subgraph. We set the length of arc $(33, t)$ to its lower bound and other arcs to their upper bounds. The resulting graph

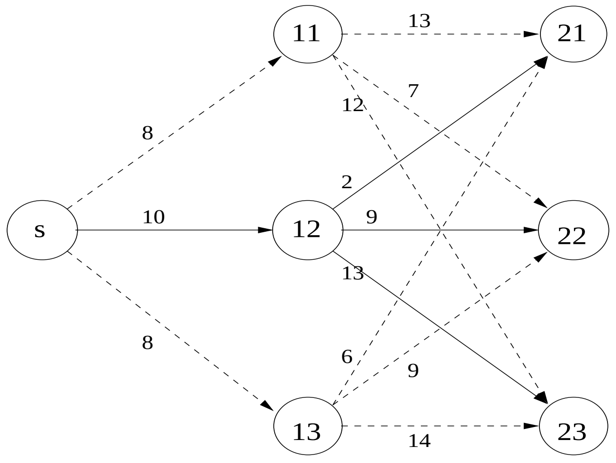

Figure 2.3: Subgraph generated by node s, nodes in layer 1 and layer 2

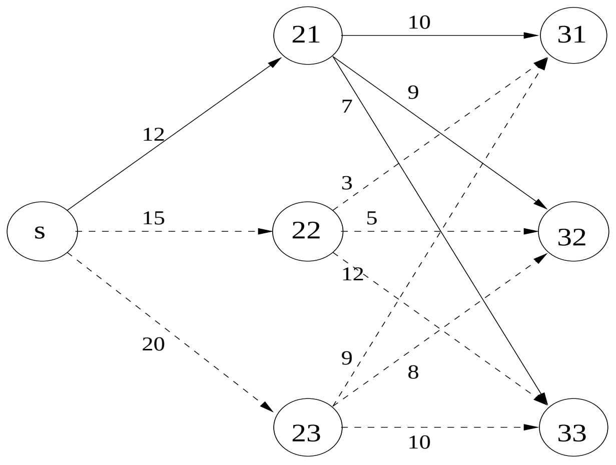

Figure 2.4: Subgraph shrunk between node s and nodes in layer 2

can be seen in figure 2.5.

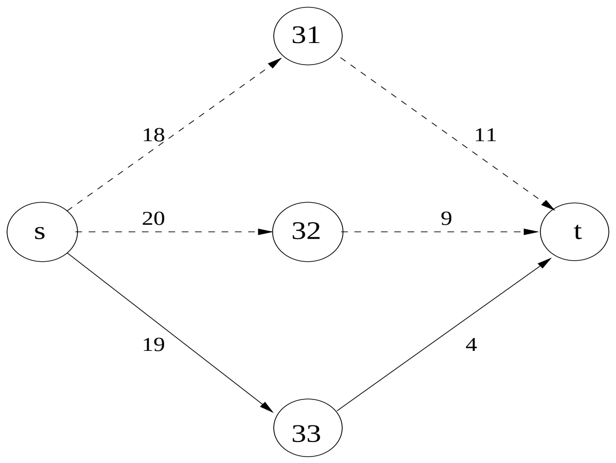

Figure 2.5: Subgraph shrunk between node s and nodes in layer 3

Finally, the shortest path in the resulting graph uses arc $(s, 12)$. By applying above procedure, we construct a weak path that uses arc $(s, 12)$. So, we can conclude that arc $(s, 12)$ is a weak arc.

Intermediate Arcs

We now consider the arcs in a layered graph other than arcs incident at nodes s and t. We call such arcs *intermediate* arcs and give a necessary condition for an intermediate arc to be weak. We can only decide non-weak arcs that does not satisfy this necessary condition. Then, we present the slightly modified version of the above procedure which decides non-weak arcs.

Proposition 2.7 *If an intermediate arc $(i1, j1)$ is weak, then it is weak in the subgraph generated by node s, nodes in layer 1 up to layer i and node $j1$, and it is weak in the subgraph generated by node $i1$, nodes in layer j up to layer m and node t.*

If an intermediate arc is not weak in the subgraph, then it cannot be weak in the whole graph. So, based on Proposition 2.7, we can check whether an arc

$(i1, j1)$ is weak or not by checking whether it is weak in the subgraph generated by node s, nodes in layer 1 up to layer i and node $j1$, and it is weak in the subgraph generated by node $i1$, nodes in layer j up to layer m and node t. The procedure is same as the above procedure except that we need to consider the whole graph.

In order to check whether arc $(i1, j1)$ is weak or not in the subgraph generated by node $i1$, nodes in layer j up to layer m and node t, we assume the origin node is $i1$ and apply the above procedure. The difference is that after the procedure, we can only decide non-weak arcs. To be more precise, we apply the procedure on the previous figure 2.2 for arc $(11, 21)$.

Example 2

To check whether arc $(11, 21)$ is weak or not, we generate the subgraph by node 11, nodes in layer 2 and layer 3, and node t. First, we need to only consider node 11, nodes in layer 2 and layer 3. We set the lengths of arcs $(11, 21)$, $(21, 31)$, $(21, 32)$, and $(21, 33)$ to their lower bounds and the lengths of the other arcs to their upper bounds. The resulting graph can be seen in figure 2.6

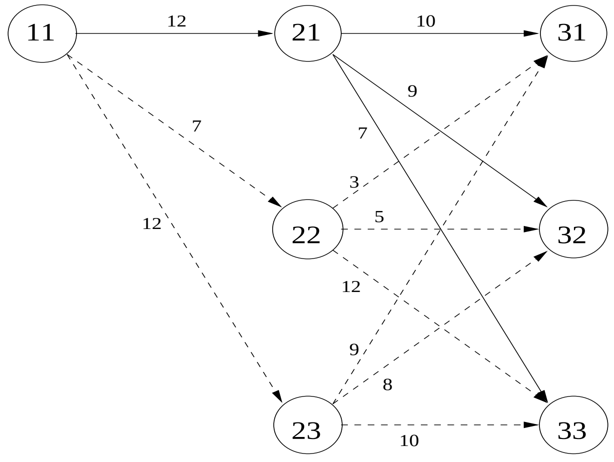

Figure 2.6: Subgraph generated by node 11, nodes in layer 2 and layer 3

The shortest path from node 11 to node 33 uses arc $(11, 21)$. In fact, we have two equal length paths but we favor the path that uses arc $(11, 21)$. Then, we shrink the graph between node 11 and nodes in layer 3 and add the node t to our subgraph. We set the length of arc $(33, t)$ to its lower bound and lengths of the other two arcs to their upper bounds. The final graph can be seen in figure 2.7.

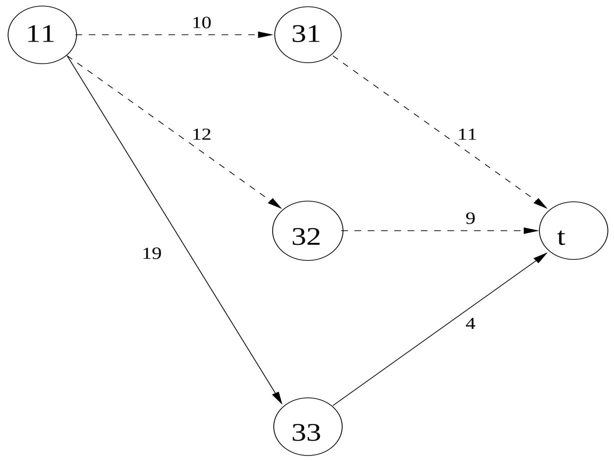

Figure 2.7: Subgraph shrunk between node 11 and nodes in layer 3

Finally, the shortest path in the resulting subgraph does not use arc $(11, 21)$. So, the necessary condition is not satisfied and we decide that arc $(11, 21)$ is not a weak arc.

Chapter 3

Computational Results

In the previous chapter, we have seen that the relative robust version of the shortest path problem is much harder than the conventional shortest path problem. It is difficult to find a polynomial time algorithm for solving it. However, for practical purposes, especially when the number of variables is large, reducing the solution space becomes an important issue. If we can identify variables that can not be candidates for a robust solution, we can eliminate these variables from the problem. So, the resulting problem will have less number of variables, and can be solved faster than the original one.

In order to solve the relative robust path problem, our approach is as follows: We have identified paths that are shortest for all realizations of arc lengths and that are shortest for some realizations of arc lengths. We called these paths permanent paths and weak paths, respectively. Then, we have shown the basic result that a relative robust path is a weak path. Therefore, in solving the problem, we need to consider only the weak path set that can possibly be robust paths. We can eliminate arcs from the problem that are proved not to be on weak paths. The resulting problem will be easier to solve.

We have conducted extensive computational studies to test the efficiency of our approach in solving the relative robust path problem. We first solved the problem with all the arcs in the graph. Then, we used procedures that are presented in chapter 2 (procedures for layered graphs) to eliminate non-weak arcs from the problem and then resolve it. The performance measure we used in comparing the efficiency of our approach is the **cpu** time.

The graphs we used in our computational experiments are layered graphs. This is to avoid shortest paths that passes through a small number of nodes, e.g. a shortest path directly from the origin node to the destination node.

The input data to relative robust shortest path problem are arc lengths, i.e., upper and lower bounds. We generate the input data as follows. We first generate a base case scenario for a given arc. We consider two different base cases randomly generated from a uniform distribution between numbers: $U(1,20)$ and $U(1,100)$. Let c_a^0 denote the value of the base case scenario. Then, the lower bounds l_a are randomly generated from a uniform distribution $U((1-d)c_a^0), (1+d)c_a^0)$ where d is a prespecified number ($0 < d < 1$). Then, the upper bounds are generated from $U(l_a + 1, (1+d)c_a^0)$.

We did our computational results on layered graphs with width 2, 3, and 5. For each set of data (number of nodes in the graph, percentage deviation from base case d), 10 problems are solved and average performance for various measures are reported. For the problem set at the fifth row of each table, we run only 5 problems. We first generate the base case from a uniform distribution $U(1,20)$ and then from a uniform distribution $U(1,100)$.

Computational studies were conducted with the use of a C code and run on a Sun workstation by using Cplex linear optimizer 5.0. The number of arcs and the number of non-weak arcs in the graph are reported to compare the numbers with different percentage deviation from base case. They are denoted as **arcs** and **non-weak**, respectively. We report three different CPU seconds for obtaining the optimal relative robust solution. By **preprocessing**, we present the time spent by preprocessing procedures. The second one, **cpu1**, corresponds the solution time of the problem with all the arcs in the graph and the third one, **cpu2**, corresponds the solution time of the problem after preprocessing. In addition, we report the percent reduction obtained from our approach, i.e., preprocessing of the graph. We first compute the difference between the cpu time of the solution with the all arcs and cpu time of the preprocessing plus cpu time of the solution after preprocessing. Then, the percent reduction is computed by taking the ratio of this difference to the cpu time of the solution before preprocessing.

We present the computational studies in two different classes of graphs: Complete layered graphs and sparse layered graphs.

3.1 Computational Results in Complete Layered Graphs

A complete layered graph is the one where each pair of distinct nodes between s and V_1, V_m and t, and V_k and V_{k+1} for $k = 1, 2, \dots, m-1$ is joined by an arc. Tables 3.1 through 3.14 show the computational results for the relative robust path problem with 30 through 420 nodes and the deviation parameter 0.3, 0.6 and 0.9, respectively. Since the solution time of the problem depends strongly on the deviation parameter from base case, we consider different sizes of graphs in presenting the computational results.

To be more precise, in Table 3.1, we conduct the experiments in a layered graph of width 2. We generate the base case scenario c_a^0 from a uniform distribution $U(1, 20)$. Then, we generate lower bounds from the uniform distribution $U((1-d)c_a^0), (1+d)c_a^0)$ and generate upper bounds from $U(l_a + 1, (1+d)c_a^0)$. For example, in first row we take a graph of 180 nodes and generate the lower bounds l_a from $U((0.7)c_a^0, (1.3)c_a^0)$, then generate the upper bounds from $U(l_a + 1, (1.3)c_a^0)$. There exist totally 360 arcs in the graph for which 176 of them were decided to be non-weak by procedures presented in Chapter 2. The time spent by preprocessing procedures is 1.12 cpu seconds. The average solution time without preprocessing of the graph takes 7.62 cpu seconds whereas it takes 2.75 cpu seconds after preprocessing. Finally, we have a %49 reduction in solution time of the problem if we preprocess the graph.

node	d	**arcs**	**non-weak**	**preprocessing**	**cpu1**	**cpu2**	**% reduction**
180	0.3	360	176	1.12	7.62	2.75	% 49
210	0.3	420	203	1.70	27.33	13.61	% 44
240	0.3	480	233	2.60	281.3	74.96	% 72
270	0.3	540	264	3.41	199.2	75.85	% 60
300	0.3	600	290	4.55	1948	352.1	% 82

Table 3.1: Computational results for base case $(1, 20)$ in a layered graph of width 2 for $d = 0.3$

node	d	**arcs**	**non-weak**	**preprocessing**	**cpu1**	**cpu2**	**% reduction**
120	0.6	240	88	0.39	5.76	2.78	% 45
150	0.6	300	111	0.73	24.09	10.27	% 54
180	0.6	360	132	1.21	233.8	78.97	% 66
210	0.6	420	154	1.85	481.7	323.4	% 32
240	0.6	480	174	2.86	506.0	256.7	% 49

Table 3.2: Computational results for base case $(1, 20)$ in a layered graph of width 2 for $d = 0.6$

node	d	**arcs**	**non-weak**	**preprocessing**	**cpu1**	**cpu2**	**% reduction**
30	0.9	60	12	0.01	0.08	0.07	% 0
60	0.9	120	21	0.07	1.27	1.04	% 13
90	0.9	180	31	0.21	19.11	15.75	% 16
120	0.9	240	44	0.43	106.6	70.19	% 34
150	0.9	300	52	0.81	1906	1062	% 44

Table 3.3: Computational results for base case $(1, 20)$ in a layered graph of width 2 for $d = 0.9$

node	d	**arcs**	**non-weak**	**preprocessing**	**cpu1**	**cpu2**	**% reduction**
240	0.3	480	273	2.60	6.56	1.91	% 31
270	0.3	540	310	3.23	10.82	5.97	% 15
300	0.3	600	339	4.55	32.89	11.02	% 53
330	0.3	660	379	5.79	111.0	20.0	% 77
360	0.3	720	413	7.68	149.1	38.9	% 69

Table 3.4: Computational results for base case $(1, 100)$ in a layered graph of width 2 for $d = 0.3$

node	d	arcs	non-weak	preprocessing	cpu1	cpu2	% reduction
180	0.6	360	168	1.24	7.42	2.48	% 50
210	0.6	420	196	1.68	29.72	9.35	% 63
240	0.6	480	223	2.86	122.8	31.16	% 72
270	0.6	540	253	3.46	370.4	86.1	% 76
300	0.6	600	280	5.07	1060	335.1	% 68

Table 3.5: Computational results for base case $(1, 100)$ in a layered graph of width 2 for $d = 0.6$

node	d	arcs	non-weak	preprocessing	cpu1	cpu2	% reduction
120	0.9	240	82	0.43	5.23	2.87	% 37
150	0.9	300	104	0.74	39.63	20.64	% 46
180	0.9	360	124	1.40	344.5	135.1	% 60
210	0.9	420	149	1.83	1600	722.1	% 55
240	0.9	480	159	3.24	6683	3242	% 51

Table 3.6: Computational results for base case $(1, 100)$ in a layered graph of width 2 for $d = 0.9$

node	d	arcs	non-weak	preprocessing	cpu1	cpu2	% reduction
180	0.3	537	323	1.95	6.28	1.23	% 49
210	0.3	627	382	2.97	4.46	1.12	% 8
240	0.3	717	433	4.37	18.42	3.74	% 56
270	0.3	807	496	6.02	31.53	5.03	% 65
300	0.3	897	558	7.76	79.64	13.95	% 73

Table 3.7: Computational results for base case $(1, 20)$ in a layered graph of width 3 for $d = 0.3$

node	d	arcs	non-weak	preprocessing	cpu1	cpu2	% reduction
180	0.6	537	222	2.06	45.25	22.48	% 46
210	0.6	627	274	3.45	291.9	102.8	% 64
240	0.6	717	299	4.49	309.3	163.4	% 46
270	0.6	807	358	7.12	354.2	180.3	% 47
300	0.6	897	393	9.32	2275	928.9	% 59

Table 3.8: Computational results for base case $(1, 20)$ in a layered graph of width 3 for $d = 0.6$

node	d	**arcs**	**non-weak**	**preprocessing**	**cpu1**	**cpu2**	**% reduction**
60	0.9	177	33	0.13	1.91	1.42	% 19
90	0.9	267	59	0.41	9.99	8.36	% 12
120	0.9	357	70	0.79	117.8	102.8	% 12
150	0.9	447	87	1.63	1946	1654	% 15
180	0.9	537	106	2.17	6785	4743	% 30

Table 3.9: Computational results for base case $(1, 20)$ in a layered graph of width 3 for $d = 0.9$

node	d	**arcs**	**non-weak**	**preprocessing**	**cpu1**	**cpu2**	**% reduction**
180	0.9	537	211	2.17	61.53	33.11	% 43
210	0.9	627	255	3.57	378.6	225.6	% 39
240	0.9	717	291	5.03	1073	386.2	% 64
240	0.9	807	338	7.26	1583	580.1	% 63
300	0.9	897	372	11.23	3977	1979	% 50

Table 3.10: Computational results for base case $(1, 100)$ in a layered graph of width 3 for $d = 0.9$

node	d	**arcs**	**non-weak**	**preprocessing**	**cpu1**	**cpu2**	**% reduction**
300	0.3	1485	1016	12.85	39.83	4.29	% 57
330	0.3	1635	1137	18.02	24.03	3.81	% 9
360	0.3	1785	1235	22.79	62.21	10.36	% 47
390	0.3	1935	1331	28.50	78.36	12.11	% 48
420	0.3	2085	1436	35.36	326.6	52.58	% 73

Table 3.11: Computational results for base case $(1, 20)$ in a layered graph of width 5 for $d = 0.3$

node	d	**arcs**	**non-weak**	**preprocessing**	**cpu1**	**cpu2**	**% reduction**
210	0.6	1035	504	6.40	12.75	6.60	-
240	0.6	1185	577	9.19	21.54	12.69	-
270	0.6	1335	639	12.82	124.8	38.95	% 59
300	0.6	1485	700	16.37	241.4	110.43	% 47
330	0.6	1635	784	22.63	374.2	159.9	% 51

Table 3.12: Computational results for base case $(1, 20)$ in a layered graph of width 5 for $d = 0.6$

node	d	**arcs**	**non-weak**	**preprocessing**	**cpu1**	**cpu2**	**% reduction**
120	0.9	585	135	1.74	15.47	12.82	% 6
150	0.9	735	161	2.97	35.97	25.40	% 21
180	0.9	885	182	5.09	167.4	120.1	% 25
210	0.9	1035	213	7.80	799.1	649.6	% 18
240	0.9	1185	288	12.11	930	611.5	% 33

Table 3.13: Computational results for base case $(1, 20)$ in a layered graph of width 5 for $d = 0.9$

node	d	**arcs**	**non-weak**	**preprocessing**	**cpu1**	**cpu2**	**% reduction**
210	0.9	1035	470	6.47	21.56	12.77	% 11
240	0.9	1185	521	9.57	76.52	37.16	% 39
270	0.9	1335	601	12.96	98.29	55.06	% 31
300	0.9	1485	658	21.31	148.5	66.79	% 41
330	0.9	1635	729	23.01	753.9	361.1	% 49

Table 3.14: Computational results for base case $(1, 100)$ in a layered graph of width 5 for $d = 0.9$

The computational results support our claim for computational efficiency of the preprocessing of graphs. In the average, the percent reduction obtained from preprocessing is %42.91. We now give a detailed analysis of percent reduction based on the factors percent deviation d from base case, width of the graph and base case distribution.

d	**% reduction**
0.3	% 51.9
0.6	% 49.5
0.9	% 32.6

Table 3.15: Deviation parameter vs. % Reduction

It can be seen from Table 3.15 that the percent reduction is higher when the deviation parameter is lower. This can be explained as follows. While the deviation parameter increases, the gap between upper and lower bounds also increases. This results into larger intervals for arc lengths. As the intervals become larger, more arc realizations are possible. Therefore we have a larger weak set, hence a smaller number of eliminated arcs from preprocessing. This in turn yields a low percent reduction from preprocessing.

In addition, it can be seen from the tables that the decision making process itself becomes harder when the deviation parameter increases. Hence, for an easier decision making process, the lower and the upper bounds should be closer to each other.

width	**% reduction**
2	% 49.3
3	% 43.0
5	% 33.3

Table 3.16: Width vs. % Reduction

When we compare the percent reduction among different width sizes of the graphs, in Table 3.16, we can see that there is a decrease in percent reduction as the width of the graph increases. This can be explained as follows. For a layered graph of width two, for any node, we have two incoming arcs and two outgoing arcs. If one of the arcs decided to be non-weak by the procedures and eliminated from the problem, in a branch-and-bound procedure, we have %50 gain in the branching tree. However, for larger widths of graphs, this gain will decrease since there are more entering and exiting arcs. Hence, the overall reduction in this case will be lower.

base case	**% reduction**
$U(1,20)$	% 39.0
$U(1,100)$	% 49.9

Table 3.17: Base case vs. % Reduction

Finally, from the comparison between two different base case distributions, in Table 3.17, we have a higher percent reduction in the base case $U(1,100)$ then in the base case $U(1,20)$, since the number of eliminated arcs in the base case $U(1,100)$ is greater then the number of eliminated arcs in the base arcs $U(1,20)$.

In addition, it can be inferred from the experimental results that as the number of nodes increases in a graph, the percent reduction we obtained from preprocessing also increases. So, the preprocessing procedures becomes a must for the graphs with larger number of nodes.

3.2 Computational Results in Sparse Layered Graphs

We now present our computational results in sparse layered graphs. A sparse layered graph is the one where each pair of distinct nodes between s and V_1, V_m and t, and V_k and V_{k+1} for $k = 1, 2, \ldots, m-1$ is joined by an arc with some probability. We consider two different probabilities: 0.75 and 0.50. Tables 3.18 through 3.26 show the computational results for the relative robust path problem for probability 0.75 with 60 through 450 nodes and the deviation parameter 0.3, 0.6 and 0.9, respectively.

node	d	**arcs**	**non-weak**	**preprocessing**	**cpu1**	**cpu2**	**% reduction**
330	0.3	533	192	10.69	27.50	9.04	% 28
360	0.3	582	213	13.69	62.24	24.97	% 38
390	0.3	629	217	17.41	652.8	235.9	% 61
420	0.3	676	250	21.93	724.1	272.1	% 59
450	0.3	731	292	27.81	2433	310	% 86

Table 3.18: Computational results for base case $(1, 20)$ in a layered graph of width 2 for $d = 0.3$, for $p = 0.75$

node	d	**arcs**	**non-weak**	**preprocessing**	**cpu1**	**cpu2**	**% reduction**
180	0.6	290	76	1.83	11.09	10.03	-
210	0.6	341	95	2.81	62.03	31.11	% 45
240	0.6	390	107	4.37	108.7	70.75	% 31
270	0.6	430	131	5.85	143.8	73.27	% 45
300	0.6	471	154	7.16	3126	1556	% 50

Table 3.19: Computational results for base case $(1, 20)$ in a layered graph of width 2 for $d = 0.6$, for $p = 0.75$

Tables 3.27 through 3.29 show the computational results for the relative robust path problem for probability 0.5 with 120 through 300 nodes and the deviation parameter 0.9.

In the average, the percent reduction obtained from preprocessing is %28.5 in sparse layered graphs. The following tables will show an analysis of percent reduction based on the factors percentage deviation and width of the graph.

node	d	**arcs**	**non-weak**	**preprocessing**	**cpu1**	**cpu2**	**% reduction**
60	0.9	97	12	0.08	0.30	0.22	% 0
90	0.9	144	20	0.27	3.03	2.41	% 11
120	0.9	192	28	0.57	5.74	4.48	% 12
150	0.9	243	37	1.14	132.8	108.5	% 17
180	0.9	290	40	1.95	154.5	116.7	% 23

Table 3.20: Computational results for base case $(1, 20)$ in a layered graph of width 2 for $d = 0.9$, for $p = 0.75$

node	d	**arcs**	**non-weak**	**preprocessing**	**cpu1**	**cpu2**	**% reduction**
300	0.3	690	316	11.79	36.25	6.73	% 49
330	0.3	756	346	16.43	44.40	13.32	% 33
360	0.3	830	381	21.74	80.66	23.76	% 44
390	0.3	897	418	28.35	190.0	28.36	% 71
420	0.3	967	464	34.13	362.0	93.92	% 65

Table 3.21: Computational results for base case $(1, 20)$ in a layered graph of width 3 for $d = 0.3$, for $p = 0.75$

node	d	**arcs**	**non-weak**	**preprocessing**	**cpu1**	**cpu2**	**% reduction**
150	0.6	341	113	1.66	3.36	2.44	-
180	0.6	411	133	3.15	16.39	9.58	% 22
210	0.6	481	167	4.57	38.26	19.46	% 37
240	0.6	550	182	6.98	289.9	174.8	% 37
270	0.6	616	186	9.98	693.1	292.9	% 56

Table 3.22: Computational results for base case $(1, 20)$ in a layered graph of width 3 for $d = 0.6$, for $p = 0.75$

node	d	**arcs**	**non-weak**	**preprocessing**	**cpu1**	**cpu2**	**% reduction**
60	0.9	139	22	0.15	0.99	0.75	% 8
90	0.9	206	32	0.48	3.34	3.00	-
120	0.9	272	42	0.99	9.38	7.75	% 7
150	0.9	343	55	1.86	206.8	159.0	% 22
180	0.9	411	66	3.23	1498	1105	% 26

Table 3.23: Computational results for base case $(1, 20)$ in a layered graph of width 3 for $d = 0.9$, for $p = 0.75$

node	d	**arcs**	**non-weak**	**preprocessing**	**cpu1**	**cpu2**	**% reduction**
300	0.3	1129	641	19.32	10.60	3.19	-
330	0.3	1230	669	27.66	87.59	8.89	% 58
360	0.3	1365	744	35.76	52.37	16.83	-
390	0.3	1454	783	44.10	73.88	16.82	% 18
420	0.3	1586	850	58.67	1229	323.9	% 69

Table 3.24: Computational results for base case $(1, 20)$ in a layered graph of width 5 for $d = 0.3$, for $p = 0.75$

node	d	**arcs**	**non-weak**	**preprocessing**	**cpu1**	**cpu2**	**% reduction**
210	0.6	785	295	8.06	16.89	9.06	-
240	0.6	898	310	12.31	29.86	12.30	% 17
270	0.6	1014	353	17.36	155.3	85.35	% 34
300	0.6	1128	404	22.78	460.6	251.9	% 40
330	0.6	1238	468	30.33	452.7	190.1	% 51

Table 3.25: Computational results for base case $(1, 20)$ in a layered graph of width 5 for $d = 0.6$, for $p = 0.75$

node	d	**arcs**	**non-weak**	**preprocessing**	**cpu1**	**cpu2**	**% reduction**
120	0.9	442	76	1.92	8.53	5.97	% 7
150	0.9	556	86	3.42	9.08	10.77	-
180	0.9	671	110	6.04	58.47	50.91	% 3
210	0.9	784	113	9.49	352.9	290.2	% 15
240	0.9	897	129	13.31	640.9	595.1	% 5

Table 3.26: Computational results for base case $(1, 20)$ in a layered graph of width 5 for $d = 0.9$, for $p = 0.75$

node	d	**arcs**	**non-weak**	**preprocessing**	**cpu1**	**cpu2**	**% reduction**
180	0.9	249	27	1.55	36.59	20.25	% 40
210	0.9	289	31	2.57	131.5	71.01	% 44
240	0.9	335	36	3.78	185.5	106.4	% 41
270	0.9	368	37	5.60	356.1	325.8	% 7
300	0.9	411	41	6.86	1270	802	% 36

Table 3.27: Computational results for base case $(1, 20)$ in a layered graph of width 2 for $d = 0.9$, for $p = 0.5$

node	d	arcs	non-weak	preprocessing	cpu1	cpu2	% reduction
180	0.9	315	41	2.42	30.51	22.83	% 17
210	0.9	364	49	3.81	118.9	83.10	% 27
240	0.9	417	71	5.51	269.1	211.7	% 19
270	0.9	466	78	8.29	3380	2211	% 34
300	0.9	530	84	10.68	5601	3546	% 36

Table 3.28: Computational results for base case $(1, 20)$ in a layered graph of width 3 for $d = 0.9$, for $p = 0.5$

node	d	arcs	non-weak	preprocessing	cpu1	cpu2	% reduction
180	0.9	465	74	3.91	38.95	33.97	% 3
210	0.9	542	77	6.04	120.2	98.35	% 13
240	0.9	616	85	8.84	849.6	574.3	% 31
270	0.9	704	98	13.13	1062	732.0	% 30
300	0.9	781	109	16.21	2075	1824	% 11

Table 3.29: Computational results for base case $(1, 20)$ in a layered graph of width 5 for $d = 0.9$, for $p = 0.5$

d	% reduction
0.3	% 45.27
0.6	% 31.0
0.9	% 18.9

Table 3.30: Deviation parameter vs. % Reduction

It can be seen from Table 3.30 that the percent reduction is higher when the deviation parameter is lower. This is because we have a larger weak set when the deviation parameter is higher. Hence a smaller number of eliminated arcs from preprocessing. This in turn yields a low percent reduction from preprocessing. When we compare the result with complete layered graphs, we see that the percent reduction is higher in complete layered graphs, since the percentage of non-weak arcs is higher in complete layered graphs.

width	**% reduction**
2	% 33.7
3	% 30.5
5	% 21.35

Table 3.31: Width vs. % Reduction

It can be seen from Table 3.31 that as the width of the graph increases, the percent reduction decreases. This can be explained by the same reasoning for complete layered graphs. The results corresponding to complete layered graphs are better than these results, since the percentage of non-weak arcs is higher in complete graphs.

In summary, the following observations can be made from the computational results:

- The preprocessing of graphs helps us significantly in solving the relative robust path problem, especially when the number of nodes is large.
- The percentage of the weak arcs in the graph depends on the interval lengths. The larger the intervals, the larger the number of weak arcs is and the lower the eliminated arcs from preprocessing. This in turn makes the percent reduction obtained from preprocessing lower.
- The decision making process itself becomes significantly harder when the gap between lower and upper bounds increases. So, for an easier decision making process, the lower and the upper bounds should be close to each other.
- As the width of the graph increases, the percent reduction decreases.

Chapter 4

Conclusion

In this thesis, we investigated the well-known shortest path problem with interval data. In order to handle uncertainty in the decision making process, we adopted a robustness approach to the problem. The robustness criteria we used was minimax and minimax regret. We saw that the problem is easily solvable under the minimax criterion. However, it is difficult to find a polynomial time algorithm in order to solve it under minimax regret criterion.

Since arc lengths are intervals, being a shortest path depends on the realizations of arc lengths. Based on these realizations, we defined permanent and weak paths. A permanent path is a shortest path for all realizations of arc lengths while a weak path is a shortest path for at least one realization. In order to find these solutions, we only considered the extreme point scenarios, i.e., the scenarios where the input data are at their lower and upper bounds.

Use of the interval data to represent uncertainty in the decision model yielded more analysis and stronger results in characterizing the structural properties of robust solutions. We saw that it is enough to consider extreme point scenarios to find the worst case scenarios of robust solutions.

Another important result was that robust solutions are weak solutions. Therefore, knowing which arcs are non-weak, we can preprocess a given graph for robust path problems. Computational results showed that the preprocessing of graphs is an efficient method in solving robust path problems, especially when the number of nodes is large.

It can be seen from computational results that the size of the weak solution set depends on the width of the intervals. As the intervals become larger, more arc realizations are possible. Therefore, we have a larger weak set. This in turn makes the percent reduction obtained from preprocessing lower. In addition, as the gap between lower and upper bounds increases, the computational effort in obtaining the solution also increases. Hence, for an easier decision making process, the lower and the upper bounds should be close to each other.

Bibliography

[1] Ahuja, R.K., T.L. Magnanti and J.B. Orlin, *Network Flows*, Prentice Hall, New Jersey, 1993.

[2] Averbakh, I., "On the Complexity of a Class of Combinatorial Optimization Problems with Uncertainty", *Mathematical Programming*, 90, 2, 263-272, 2001.

[3] Daniles, R.L., and P. Kouvelis, "Robust Scheduling to Hedge Against Processing Time Uncertainty in Single-Stage Production", *Management Science*, 41, 2, 363-376, 1995.

[4] Demir, M.H., B.Ç. Tansel and G.F. Scheuenstuhl, "Tree Network 1-Median Location with Interval Data: A Parameter Space Approach", To be appear in *IEE Transactions*, 2001.

[5] Dijkstra, E.W., "A Note on Two Problems in Connection With Graphs", *Numerische Mathematik*, 1, 269-271, 1959.

[6] Gupta, S.K., and J. Rosenhead, "Robustness in Sequential Investment Decisions", *Management Science*, 15, 2, 18-29, 1972.

[7] Kouvelis, P., A.A. Karawarwala and G.J. Guiterez, "Algorithms for Robust Single and Multiple Period Layout Planning for Manufacturing Systems", *European Journal of Operations Research*, 63, 287-303, 1992.

[8] Kouvelis, P. and G. Yu, *Robust Discrete Optimization and Its Applications*, Kluwer Academic Publishers, Netherlands, 1997.

[9] Rosenhead, M.J., M. Elton and S.K. Gupta, "Robustness and Optimality as Criteria for Strategic Decisions", *Operational Research Quarterly*, 23, 4, 413-430, 1972.

[10] Sengupta, J.K., "Robust Decisions in Economic Models", *Computers and Operations Research*, 18, 2, 221-232, 1991.

[11] Tansel, B.Ç. and G.F. Scheuenstuhl, "Facility Location on Tree Networks with Imprecise Data", Research Report:IEOR-8819. Department of Industrial Engineering, Bilkent University, 1988.

[12] Tüfekçi, Ö.A., "Robust Solutions to Single and Multi-Period Machine Layout Problems with Interval Flows", Master Thesis, Department of Industrial Engineering, Bilkent University, Ankara, Turkey, 1997.

[13] Yaman, H., "Essays on Some Combinatorial Problems", Master Thesis, Department of Industrial Engineering, Bilkent University, Ankara, Turkey, 1999.

[14] Yu, G. and J. Yang, "On the Shortest Path Problem", *Computers and Operations Research*, 25, 457-468, 1998.