The Robust Shortest Path Problem with Interval Data

Oya E. Karaşan*, Mustafa Ç. Pınar*, Hande Yaman†

August 6, 2001

Abstract

Motivated by telecommunication applications, we investigate the well-known shortest path problem on directed acyclic graphs under arc length uncertainties. We model data uncertainty by treating the arc lengths as interval ranges. In order to handle uncertainty in the decision making process, a robustness approach is adopted. The robustness criteria used in the paper 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. We classify arcs based on the realization of the input data. We show that knowing those arcs which are never on shortest paths we can preprocess a graph prior to solution of the 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, Interval Data, Robust Optimization
Abbreviated title. Robust Shortest Path Problem.

*Department of Industrial Engineering, Bilkent University, 06533 Bilkent, Ankara, Turkey. E-mail: *mustafap@bilkent.edu.tr, karasan@bilkent.edu.tr,*

†Université Libre de Bruxelles, Bruxelles, Belgium. *hyaman@smg.ulb.ac.be*

1 Introduction and Background

In this paper we investigate the well-known shortest path problem on directed acyclic graphs where the input data of the problem are uncertain. Uncertainty is taken here to mean that there is a range of possible realizations for each piece of data, but the actual realizations are not known. We express this range of realizations as an interval range.

The motivation for studying the shortest path problem on directed acyclic graphs with interval data comes from telecommunication problems where a communication network is used to send packets from a certain source to a certain destination. The aim is to determine a path that minimizes the total delay time where the traffic conditions at various links are uncertain due to the time varying nature of the network congestion. Since the network manager does not have full information on the link delays, it faces the a priori choice of a robust route where the total delay is acceptable regardless of the realized congestion [9].

Mathematical programming models usually have the problem of imprecise data in modelling 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 literature. A popular tool is "sensitivity analysis" which is a post-optimality tool. It 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, it may be impossible to attribute probabilities to the possible outcomes of any decision.

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. However, even if probabilities are assigned successfully, it becomes computationally very difficult to solve the problem especially with a nonlinear objective function and discrete decision variables. 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. [10], Sengupta [11], Kouvelis et al. [8]and Daniels and Kouvelis [3]) [12]".

Kouvelis and Yu [9] 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 paper, 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 require 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 [9] 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 become 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.

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 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 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 may grow 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 results in effective polynomial time procedures. We show that, knowing which arcs are never on shortest paths for any 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. Hence, 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 paper is organized as follows: In section 2, we give the formal definitions of absolute and relative shortest path problems in directed acyclic graphs with interval data. We derive a mixed integer programming formulation to find a relative robust path in a graph. Then, we make an analysis of paths and arcs, and give some basic results for robust path problems. In section 3, we present our computational results. Finally, we give conclusions in section 4.

2 Shortest Path Problem with Interval Data

In this section 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 (or, alternatively, s) and the destination node n (or, alternatively t), 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 $\Gamma^+(j) = \{k \in V : (j,k) \in A\}$. 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 of network flow problems with a single source and a single sink. An efficient $O(|V|^2)$ labeling algorithm in general networks was given by Dijkstra [5]. Other polynomial time algorithms with complexity $O(|A|)$ time can be found in [1].

Here, arc length uncertainty means 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 $\bar{l}_{ij}$ i.e., $\underline{l}_{ij} \leq l_{ij} \leq \bar{l}_{ij}$. 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 $[\underline{l}_{ij}, \bar{l}_{ij}]$. A realization of all arc lengths is called a scenario s. l^s_{ij} 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^s_p the length of path p in scenario s and $\bar{l}_p$, $\underline{l}_p$ 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.

The rest of the section 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. Consequently, in section 2.3 we make a short analysis of paths and distinguish those that are shortest for some realization. Then, we derive some basic results about robust path problems. Finally, in section 2.4, we study arcs which are never on shortest paths.

2.1 Absolute Robustness

A robust path is defined to be the one which performs satisfactorily whatever data is realized. In this section, we utilize 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 wish 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. 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 length 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 the problem:

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

This is identical to the conventional shortest path problem. Hence, the absolute robust path problem can be solved in polynomial time.

Under the absolute robustness criterion, the solutions are not sensitive 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

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 wish to find a path such that the maximum difference between the length of this path and the length of the shortest path over all realizations 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 $p^r \in \arg\min_{p \in P} \max_{s \in S} l_p^s - l_{p^*(s)}^s$ where $p^*(s)$ denotes the shortest path in scenario s.

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:

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

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 order to find a 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, namely as many as the number of paths in the graph. However, the number of paths in a graph may grow exponentially with the number of nodes in the graph.

Next, we present a mixed integer programming formulation to find a relative robust path in a graph. In the formulation, the length of arc (i,j) is defined as $l_{ij} = \underline{l}_{ij} + (\bar{l}_{ij} - \underline{l}_{ij}) y_{ij}$ 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:

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

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 relative robust path problem is as follows:

(RRP)

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

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.

Kouvelis and Yu [9] 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. They conjectured that the relative robust path problem with interval data is also NP-complete.

2.3 Paths

In the previous section, we have seen that the relative robust path problem seems to be significantly 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 undertake an analysis of paths according to the realizations of arc lengths. We characterize paths that are shortest for some realizations of arc lengths (weak paths). These concepts were defined by Demir et al. [4], and used in [7].

2.3.1 Weak Paths

We search 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 2.3. 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 2.4. *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 at 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,

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

Hence,

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

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

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

Adding a constant to both sides we have

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

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. $\square$

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 since an absolute robust path is a shortest path under the scenario in which all arc lengths are at their upper bounds. We now show that, a relative robust path is also a weak path.

Proposition 2.5. *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, i.e., the scenario which results in worst robust deviation for path p'. Then, we have:

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

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

2.4 Arcs

In this section, we classify arcs that are on shortest paths for some realizations (weak arcs), and those that are never on shortest paths (non-weak arcs). We can use this information in solving the relative robust path problem. If we can determine which arcs are never on shortest paths, we can eliminate these arcs from the graph, since a relative robust path being a weak path will never use these arcs.

Definition 2.6. An arc (i, j) is said to be a **weak arc** if it is on one of the weak paths.

Here, we 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

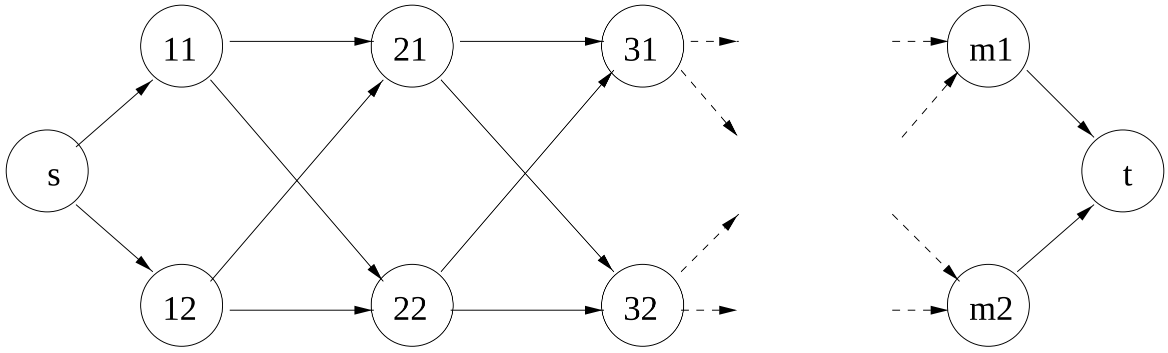

Figure 1: An m layered graph with width 2

$V = \{s\} \cup V_1 \cup V_2 \cup \ldots \cup V_m \cup \{t\}$ 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. Figure 1 shows an example of a m layered graph with width 2. However, this is not a restrictive analysis since any acyclic directed graph can be converted into a layered graph by adding dummy nodes and arcs.

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 non-weak since for intermediate arcs we only have a necessary condition. Both 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 identical but applied to the mirror version of the graph.

The logic of the procedure is to construct a weak path using arcs incident at node s if one exists. The lengths of all the 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 $2j$. 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 $(s, 11)$ (we can keep track of this by labeling the arc length in the shrunk graph to its lower bound), 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 $(s, 11)$, 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 the procedure applied to the mirror version of the graph.

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

Procedure WeakArc

1. Generate the subgraph starting from node s, nodes in V_1 and nodes in V_2.
2. Set $l_{ij} = \bar{l}_{ij}\ \forall (i,j) \in A$.
3. Set $l_{(s,11)} = \underline{l}_{(s,11)}$ and $l_{(11,jk)} = \underline{l}_{(11,jk)}$ for all jk in V_2.
4. For $b \in B = \{2, 3, \ldots, m+1\}$
   (a) Find shortest paths between node s to nodes in V_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 shortest paths that use arc $(s, 11)$, set $l_{(bj,(b+1)k)} = \underline{l}_{(bj,(b+1)k)}$ for all $k \in V_{b+1}$ $(V_{m+1} = t)$. Shrink the graph. Change the labels accordingly to keep track of all paths using $(s, 11)$.
5. If the shortest path between nodes s and t uses arc $(s, 11)$, then arc $(s, 11)$ is weak. Otherwise, it is not.

Proposition 2.7. *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. In each layer we need to find w shortest paths. In the worst case, we go through all the m layers in the graph. □

2.4.1 Example 1

To clarify the above procedure, we apply it on the graph given in Figure 2 which is a 3 layered graph with width 3. Let us pick arc $(s, 12)$.

We first generate the subgraph with nodes s, nodes in V_1 and nodes in V_2. We set the lengths of arcs $(1, 12)$, $(12, 21)$, $(12, 22)$, 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

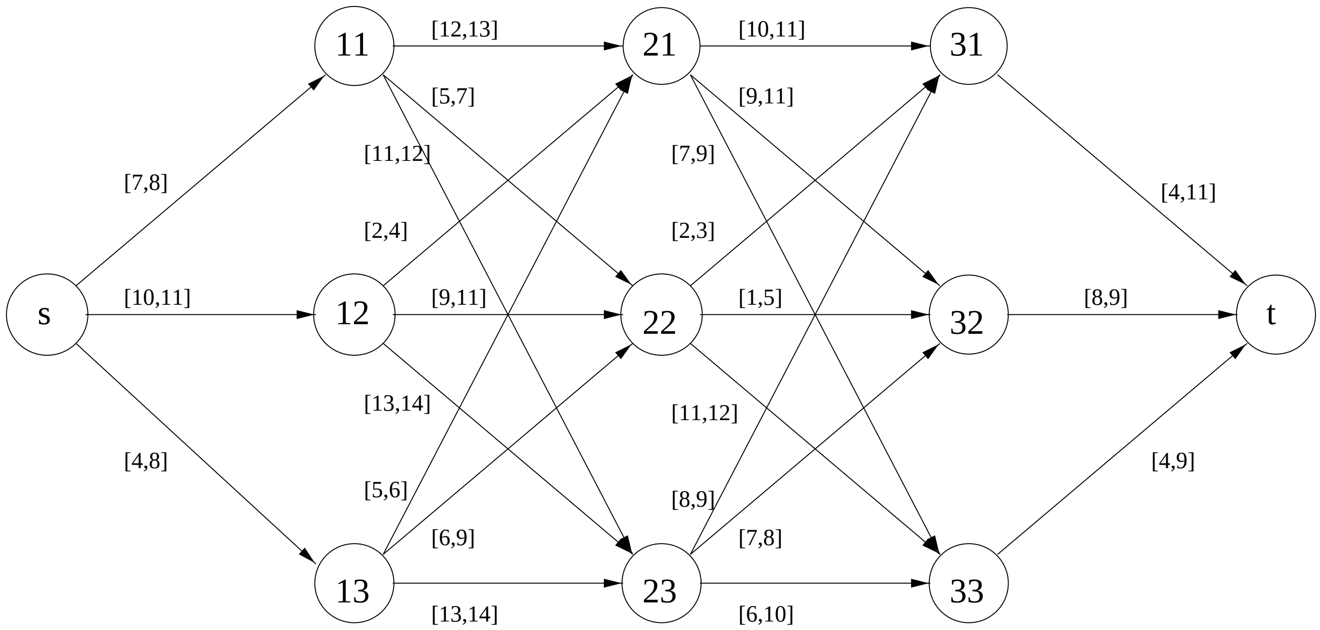

Figure 2: A 3 layered graph with width 3

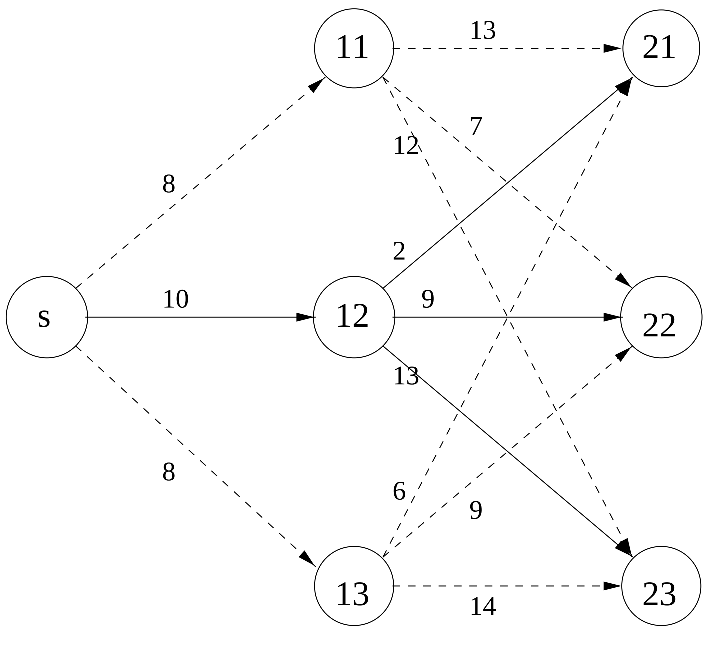

Figure 3: Subgraph generated by node s, nodes in layer 1 and layer 2

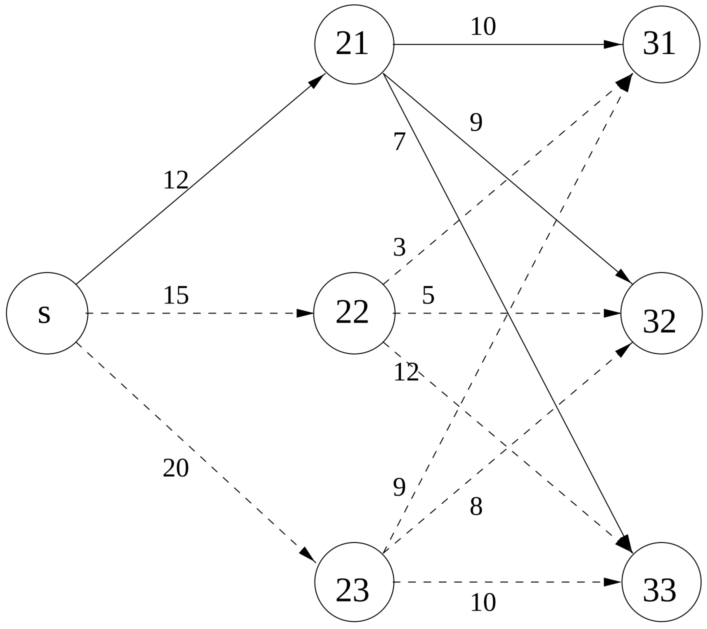

Figure 4: Subgraph shrunk between node s and nodes in layer 2

subgraph is given in Figure 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 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 $(1, 12)$. 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 can be seen in Figure 5.

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 intermediate arcs in a layered graph and give a necessary condition for an intermediate arc to be weak. We can only identify non-weak arcs that do not satisfy this necessary condition.

Proposition 2.8. *If an intermediate arc $(i1, j1)$ is a weak arc 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.*

As the proof is very simple, it is omitted.

Based on Proposition 2.8, we check whether an arc $(i1, j1)$ 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 either one of the two conditions fails to hold then we decide that the arc is non-weak. If both conditions hold, the test is non-conclusive.

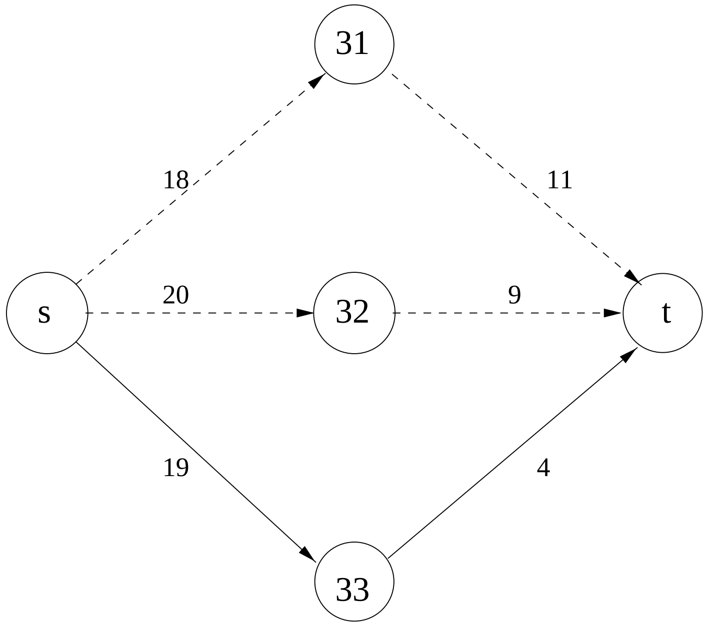

Figure 5: Subgraph shrunk between node s and nodes in layer 3

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 procedure WeakArc. For illustration we apply the procedure on the example of Figure 2 for arc $(11, 21)$.

2.4.2 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 6

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 7.

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.

In concluding this section, we note that we have a mixed-integer programming formulation to detect weak and non-weak arcs in an acyclic graph. However, since this approach is very time consuming, it was not reported in the present paper. Restricted experimentation with small graphs showed that the procedures discussed above identify most of the non-weak arcs, and hence are quite satisfactory in practice. We also have a simple but lengthy procedure that identifies all weak arcs in a layered graph of width 2 in polynomial time. However, we do not discuss this procedure as the procedures discussed above perform quite well in identifying non-weak intermediate arcs as well for layered graphs of width 2.

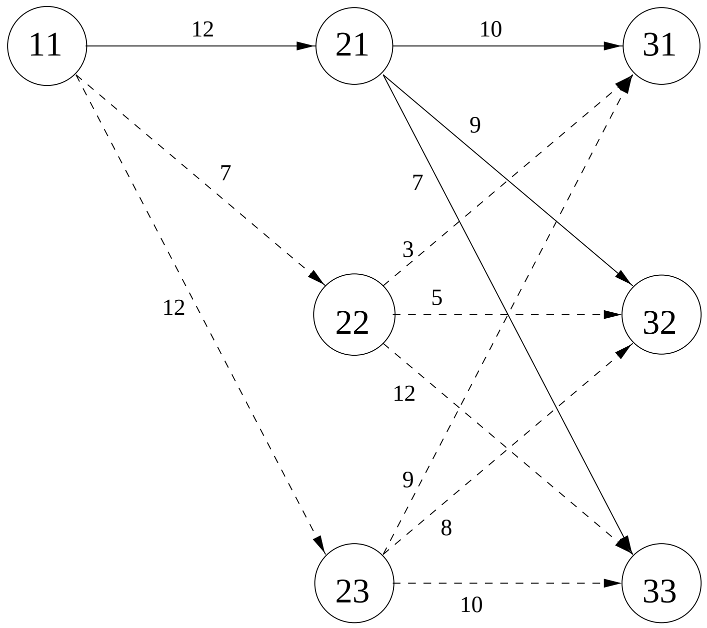

Figure 6: Subgraph generated by node 11, nodes in layer 2 and layer 3

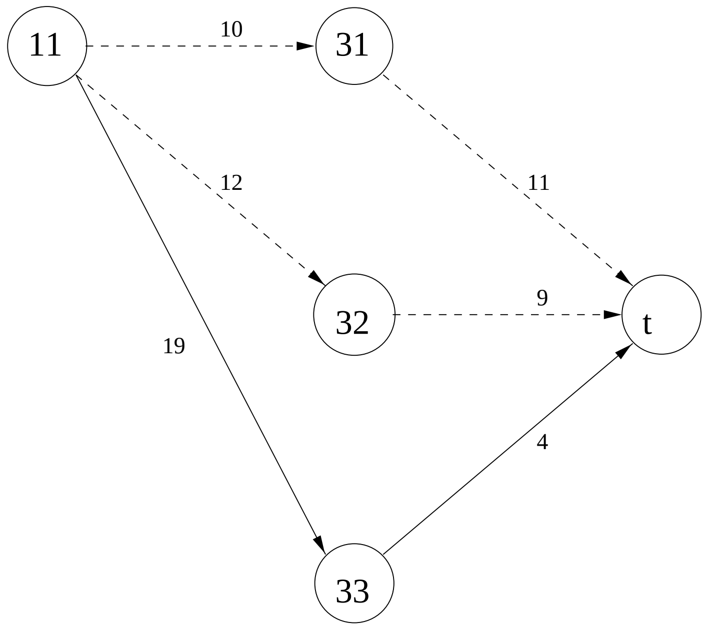

Figure 7: Subgraph shrunk between node 11 and nodes in layer 3

3 Computational Results

To solve the relative robust path problem, our approach is as follows: We have identified paths that are shortest for some realizations of arc lengths. We called these paths weak paths. 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 now 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 section 2 to eliminate non-weak arcs from the problem and then resolved it. The performance measure we used in comparing the efficiency of our approach is the **cpu** time.

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 $U(l_a + 1, (1+d)c_a^0)$.

Computational results on layered graphs with width 2, 3, and 5 are reported below. 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 as a result of long run times. 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 on a Sun workstation 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 deviations 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 to the solution time of the problem with all the arcs in the graph and the third one, **cpu2**, corresponds to 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.

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

The computational results support our claim for computational efficiency of the preprocessing of graphs. On 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. It is observed 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.

When we compare the percent reduction among different width sizes of the graphs, 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 is 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 is lower.

Finally, from the comparison between two different base case distributions, 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

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 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 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 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 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 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 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 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 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	% 0
240	0.6	1185	577	9.19	21.54	12.69	% 0
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 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 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 14: Computational results for base case $(1, 100)$ in a layered graph of width 5 for $d = 0.9$

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. Therefore, the preprocessing procedures become a must for graphs with larger number of nodes.

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.
- As the width of the graph increases, the percent reduction decreases.

4 Conclusion

Motivated by telecommunication applications, 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, the concept of a shortest path had to be revised. Based on the realizations of arc lengths, we defined weak paths that are shortest for at least one realization. It was shown that robust paths are weak paths. 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.

Acknowledgment. The authors are indebted to A.S. Karaman for conducting the computational tests, and to B. Tansel for useful discussions on the subject of this paper.

References

[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] Daniels, 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 appear in *IIE 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] Karaşan, O.E., H. Yaman and M.Ç. Pınar, The Robust Spanning Tree Problem with Interval Data, *Operations Research Letters*, 29, 31–40, 2001.

[8] 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.

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

[10] 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.

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

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