QUALITATIVE OPTIMIZATION: DEVELOPMENT OF A METHODOLOGY FOR DETERMINING THE SHORTEST PATH OF A NETWORK WITH INTERVAL-VALUED ARC LENGTHS

By

SHERRI SHEARON AVERY

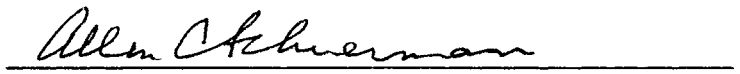
Bachelor of Arts
California State University at Fullerton
Fullerton, CA
1989

Master of Science
Trinity College
Hartford, CT
1995

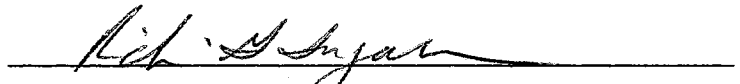
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
December, 2003

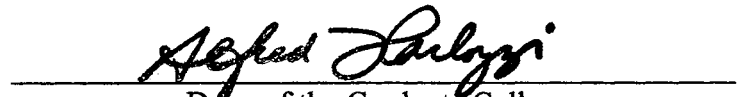
QUALITATIVE OPTIMIZATION: DEVELOPMENT OF A METHODOLOGY FOR DETERMINING THE SHORTEST PATH OF A NETWORK WITH INTERVAL-VALUED ARC LENGTHS

Thesis Approved:



Thesis Advisor





Dean of the Graduate College

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my major advisor, Dr. Allen Schuermann, for the hours of time that he has spent on my behalf. I greatly appreciate his assistance, guidance, supervision, friendship and support. My appreciation extends to my other committee members Dr. Ricki G. Ingalls, Dr. Camille DeYong, and Dr. P. Larry Claypool each of whom has given greatly needed assistance, encouragement, and advice. Specifically, I would like to thank Dr. Ricki G. Ingalls for the inspiration of this thesis topic.

Foremost, I wish to express my greatest appreciation to my parents Nathan Shearon and Jeanne Lee. They have spent my lifetime convincing me that I can complete any task that I choose. I also wish to thank my husband, Daniel, for giving me everything that I have ever needed, love and support. Additionally, I wish to thank my children, Nathan and Alyssa, for being the joy in my life.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

THE RESEARCH PROBLEM

Introduction

Many interesting and important optimization applications arise in the study of networks. Networks have been used to model science, engineering, and business applications of transportation, communication, mechanical, hydraulic, electrical and economic systems. Additionally, networks have been used to model systems based on logical connections and states of a discrete system [15].

In modeling systems as networks, solutions may be found for many problems associated with that system. Optimization techniques are used to find solutions for matching problems, assignment problems, analysis of flows, feasibility problems, routing optimization problems, critical path problems, and minimum/shortest path problems.

Traditional network applications are based on constant-valued arc costs. However, this assumption is often unrealistic and network optimization has been criticized extensively for the use of assumptions like this. A solution with a minimum number of assumptions is highly sought after and is extremely useful. Discrete event simulation may yield results that accurately describe a system without such stringent assumptions.

The simulation clock is the center of a discrete event simulation. As an event is scheduled to occur, the simulation clock advances to the time of the next event. The next event in the simulation is determined from a selection of different activities of the model. The information regarding the time between specific activities, e.g. arrivals, are generally given as a specific probability distribution, e.g. EXPO (5 min). The simulation generates

a random number from the distribution to generate the time of the next event specific to that activity. The inputs to a discrete event simulation are generally stochastic in nature and if empirical data is not available, it may be difficult to accurately define the appropriate probability distributions that describe the stochastic activities [8].

If empirical data is not available, assuming that the arc length is contained in a specific interval would reduce the number of assumptions and would generally yield a more beneficial analysis. However, interval-valued arc lengths complicate our ability to solve the problem. As the size of the problem increases, the time and effort required to obtain the solution increases exponentially [7]. In solving this type of problem, decisions must be made which depend on the value on the arc. Figure 1 shows an "ENTER-SERVICE-EXIT" model with interval-valued inputs. Assuming that one entity is already in the system, the simulation does not know the next event. The next event in the simulation could either be another entrance into the model, which will occur between 3 to 8 time units from now, or an exit which may occur between 4 to 6 time units from now. A Qualitative Discrete Event Simulation creates a "thread" for every possible next event, e.g. ENTER in [3, 4] time units, EXIT in [4, 6] time units, or ENTER in [4, 6] time units

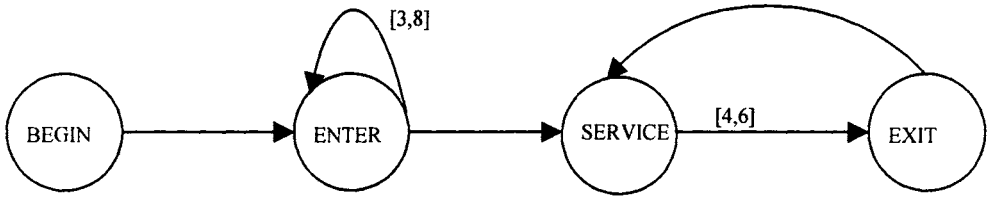


Figure 1:ENTER-SERVICE-EXIT Model

[7]. As the simulation progresses, the number of threads grows exponentially. Numerous threads must be created to appropriately describe all possible scenarios.

Qualitative Simulation is an ideal technique to analyze this problem with interval valued arc lengths [5]. Qualitative Simulation predicts all possible behaviors of a system. Qualitative Discrete Event Simulation, (QDES), is an area of study that is groundbreaking and still in its early stages of development. QDES has been used to generate threads on the model similar to Figure 1 and in PERT scheduling [7]. Ingalls has suggested that additional work is needed to appropriately analyze the output of this simulation technique [7].

Finding the shortest path of a network from an originating node to a terminating node is a well-established problem. However, this problem has not been solved with arc values that are contained in some known interval. The primary objective of this research was the development of an algorithm for the interval-valued problem that will ensure that all possible shortest paths have been generated. However, while the traditional problem is apt to have a unique shortest path, it is likely that there will be multiple solutions for the interval-valued problem. Therefore, a second objective of this research was the development of a methodology that would provide for an intelligent consolidation of the initial set of solutions. However, since it is unlikely that this reduction in the set of solutions will result in a unique path, it was useful to compare the resulting solutions in the consolidated solution set. Additionally, different decision-makers may not agree of what defines the "best" shortest path solution. Thus, a sub-objective of the second objective was to develop a methodology that would allow the evaluation of alternative attributes of the consolidated set of solutions. Each attribute of a shortest path solution

gives the decision-maker information with regard to the quality of each path given a specific objective. A decision-maker may choose one single attribute or combine the qualities of several attributes to determine the "best" among the consolidated set of solutions. Based on the decision-maker's own specific definition of "best," he/she will be able to select the "best" path from among the set of shortest paths.

The objectives of this research effort were:

1) To solve the shortest path problem with interval-valued arc lengths:
   a. Formulate a qualitative optimization algorithm using the concepts and techniques of Qualitative Discrete Event Simulation.
   b. Generate the set of all possible shortest paths.
2) To develop a methodology for the analysis of the output of the qualitative optimization:
   a. Create a combined thread set by reducing the size of the initial thread set.
   b. Identify specific attributes of the solution set.
      i. Identify non-dominated threads.
      ii. Minimize the maximum regret associated with the selection of a particular thread.
      iii. Rank the combined thread set by each thread's relative occurrence.
      iv. Identify shared sub-paths.

Problem

To appropriately review similar areas of research, some elementary graph theory notation and definitions are necessary. Definitions below are by R. T. Rockefeller [15].

- NETWORK: Two abstract sets N and A and a function that assigns each $j \in A$ to a pair (i, i') $\in \mathbf{N \times N}$. The elements of N are called NODES and are represented pictorially by small circles. The cardinality of the node set N is conventionally given as n. The elements of A are ARCS, each arc is denoted by an ordered pair (i, j), where i, j $\in$ N. Arcs are represented pictorially by arrows where the direction of the arrow shows the orientation of the arc. The cardinality of the arc set A is conventionally given as m.
- PATH: A path P in a Network G is a finite sequence i_0, (i_0, i_1), i_1,..., i_{r-1}, (i_{r-1}, i_r),i_r where each i_j is a node and (i_j, i_k) is an arc. The initial node of P is i_0 and the terminal node is i_r.
- CIRCUIT: A path in a network with the same initial node and terminal node.
- SIMPLE PATH: A simple (or elementary) path is one that uses no node more than once.
- POSITIVE PATH: A path containing only arcs a_{ij} traversed in the direction from node i to node j.
- CONNECTED: A network G is connected if for every pair of different nodes s and s', there is a path P: s $\rightarrow$ s'. (P need not be a positive path.)
- ACYCLIC: A network G is acyclic if G possesses no positive circuits.

Consider an acyclic connected network G = (N, A, C) with node set N, arc set A, and arc performance measure set C. Node set N contains nodes N = {1, 2,..., n}. Arc set A contains m arcs, a_{ij} from Node i to Node j. Associated with traversing each arc a_{ij} is a measure of performance c_{ij}, such that $c_{ij} \in [l, u]$. l is the lower bound of the performance measure associated with traversing arc a_{ij}. u is the upper bound of the performance

measure associated with traversing arc a_{ij}. Network N contains one source node, x, and one sink node, y. P(x, y) is defined as a unique positive, simple path with initial node x and terminal node y.

1) To solve the shortest path problem with interval-valued arc lengths:
   a. Formulate a qualitative optimization algorithm using the concepts and techniques of Qualitative Discrete Event Simulation.
   b. Generate the set of all possible shortest paths, P*(x, y), such that the total measure of performance of the path which is equal to $\sum_{a_{ij} \in P^*} c_{ij}$ is minimized.

2) To develop a methodology for the analysis of the output of the qualitative optimization:
   a. Create a combined thread set by reducing the size of the initial thread set.
   b. Identify specific attributes of the solution set.
      i. Identify non-dominated threads.
      ii. Minimize the maximum regret associated with the selection of a particular thread.
      iii. Rank the combined thread set by each thread's relative occurrence.
      iv. Identify shared sub-paths.

Research Contributions

Systems have been modeled and optimized using traditional network optimization techniques. The implementation of interval-valued performance measures for traversing

an arc is necessary to yield robust solutions. Figure 2 shows a graph with nodes A, B, C, and D. The enumeration of all possible paths from A to D results in the following paths:

Path 1 = Node A → Node B → Node D = 2 + 3 = 5 units.

Path 2 = Node A → Node D = 7 units.

Path 3 = Node A → Node C → Node D = 3 + 3 = 6 units [1].

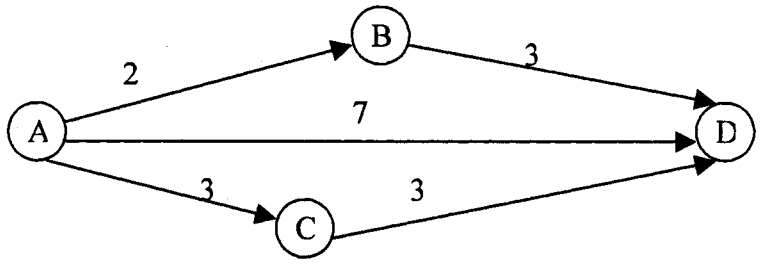


Figure 2: Transportation Network I: Nodes A-D

Path 1 is the shortest path with cumulative measure of performance of 5 units. However, variability exists in almost all realistic models. Assuming that the measures of performance of traversing arcs were as shown in Figure 3, Path 1 would not be the shortest path of the network. This trivial example does not fully show the possible problems with choosing the "wrong" optimal path. However, even a slight change to the measure of performance values along an arc can yield a dramatically different optimal path. Generally speaking, if the assumptions of the model are invalid, so-called optimal solutions can be far from optimal.

Other research has attempted to generate solutions to network optimization problems considering inherit variability innate to real-life applications. Chabini and Lan give a solution for shortest path problems in dynamic networks, in which travel times are discrete and time-dependent [3]. Okada and Soper solve the shortest path problem on a network with fuzzy arc lengths, where fuzzy numbers are a very specific type of

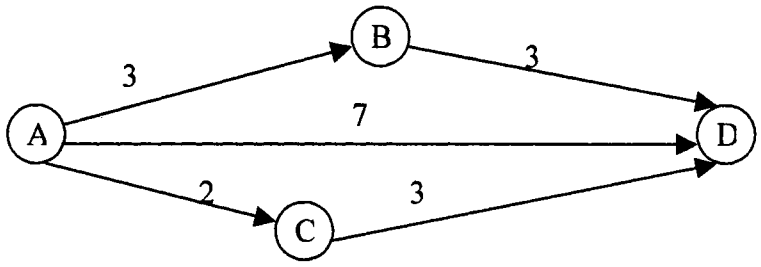


Figure 3: Transportation Network II: Nodes A-D

stochastic numbers [13]. Sudharasan has used fuzzy distances in a routing algorithm analysis for optimal web path estimation for performance measures like distance, mean packet delay and network throughput [18]. Yaman, Karasan, and Pinar analyze the robust spanning tree problem with interval data [20]. Chen and Lin use interval arc values in determining the optimum location of the 1-median of a tree [4].

Work in the area of optimizing networks that contain minimal assumptions is well-documented and an important area of research [4, 5, 13, 18, 20]. However, research has not yet been explored in finding a shortest path with interval arc values. This research effort will expand the realm of knowledge to include the formulation of an algorithm for the shortest path of a network with interval arc values. Additionally, this research effort will formulate a systematic approach for output analysis of the algorithm, which finds all shortest paths of such a network.

Research Objectives

Engineering, science and business systems have been modeled extensively as networks. These network models give the decision maker optimal information with regard to several types of problems: spanning trees, feasible flows, maximum flows, minimum path, maximum tension, assignment, matching, and shortest path [15].

Several algorithms have been developed to solve special cases of the aforementioned types of network optimizations. Networks that satisfy the assumptions of these special cases contain optimal solutions that are linear. The implementation of these algorithms is relatively straightforward due to the linear nature of the network [14]. However, these assumptions are often too restrictive and unrealistic. This research will formulate a solution to the shortest path of network without the assumption of constant values for the

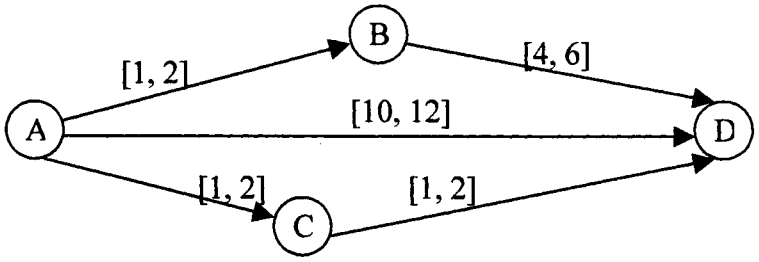


Figure 4: Transportation Network III: Nodes A-D

measure of performance for traversing an arc.

The following is a trivial example of a network. Figure 4 shows the network with nodes A, B, C, and D. The interval-valued measures of performance are shown on the arcs. The objective is to find the shortest path from the node A to node D

One alternative to find the shortest path from Node A to Node D is to find all paths from A to D. The enumeration of all possible paths from A to D results in the following paths:

Path 1 = Node A → Node B → Node D = [1, 2] + [4, 6] = [5, 8] units.

Path 2 = Node A → Node D = [10, 12] units.

Path 3 = Node A → Node C → Node D = [1, 2] + [1, 2] = [2, 4] units [1].

Therefore, Path 3 results in the unique shortest path from A to D. This is a trivial example and enumerating all possible paths is performed easily. However, enumerating

all paths of a network is not an efficient technique to find the shortest path(s). This research effort will develop a algorithm using the techniques of QDES whose output is the set of all possible shortest paths of a network. Figure 5 shows a network with the same nodes and arcs as in Figure 4. However, the measures of performance on the arcs are different. The enumeration of all possible paths from A to D results in the following paths:

Path 1 = Node A → Node B → Node D = [1, 2] + [4, 6] = [5, 8] units

Path 2 = Node A → Node D = [10, 12] units

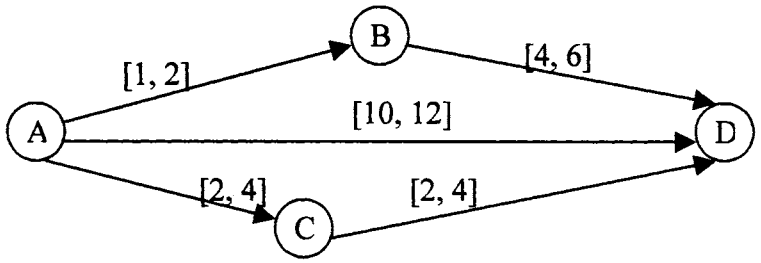


Figure 5: Transportation Network IV: Nodes A-D

Path 3 = Node A → Node C → Node D = [2, 4] + [2, 4] = [4, 8] units.

There are two possible shortest paths of the network shown in Figure 5: Path 1 or Path 3. For this network, both Path 1 and Path 3 are shortest paths depending on the exact value associated with traversing an arc.

Path 1 has a cumulative measure of performance between 5 and 8 units. Path 3 has a cumulative measure of performance between 4 and 8 units. If conditions exists such that the measure of performance of Path 3 is between 4 and 5 units then Path 3 is the shortest path. If conditions exist such that the cumulative measure of performance of Path 3 is between 5 and 8 units, Path 1 and Path 3 are both members of the set of optimal paths. Specifically for this network, Path 3 is not dominated by any other path. That is, there is

no other shortest path that can possibly yield a shorter path and Path 3 can be considered the "best" path. Again, the trivial nature of the network in Figure 5 yields a straightforward solution of an optimal path using non-dominating path analysis.

However, networks that are more complex are likely to contain numerous shortest paths in which the non-dominating analysis may not clearly select the optimal shortest path. The non-dominating path analysis is only one of the techniques of the methodology for the second objective of this research project.

The objectives of this research effort are:

1) To solve the shortest path problem with interval-valued arc lengths:
   a. Formulate a qualitative optimization algorithm using the concepts and techniques of Qualitative Discrete Event Simulation.
   b. Generate the set of all possible shortest paths, $P^*(x, y)$, such that the total measure of performance of the path which is equal to $\sum_{a_{ij} \in P^*} c_{ij}$ is minimized.

2) To develop a methodology for the analysis of the output of the qualitative optimization:
   a. Create a combined thread set by reducing the size of the initial thread set.
   b. Identify specific attributes of the solution set.
      i. Identify non-dominated threads.
      ii. Minimize the maximum regret associated with the selection of a particular thread.
      iii. Rank the combined thread set by each thread's relative occurrence.
      iv. Identify shared sub-paths.

CHAPTER 2

LITERATURE REVIEW

Introduction

Applications of the shortest path have been used extensively for transportation networks, communication networks, mechanical or electrical systems, and many more. "Shortest path problems arise both as main decision questions and as steps in other computations" [14, p. 413]. It was this research effort's primary objective to extend the applicability of the shortest path by solving the shortest path problem without the restriction of constant valued arc lengths. Due to the fact that for some applications "...neither a deterministic approach nor a stochastic approach would be appropriate" [4, p. 94], the problem shall be extended for measure of performance along the arc to be any value within an interval, i.e. an interval-valued measure of performance.

Analysis has been performed by a handful of researchers with regard to networks with non-constant arc values [4, 5, 13, 18, 20]. Networks with fuzzy arc values have been analyzed for both the shortest path and minimum spanning tree problems [13, 18]. A graph theoretical approach to the robust spanning tree problem with interval data is also in its early stages of research [20].

To fully appreciate the applicability of modeling problems as shortest paths, some fundamental graph theory concepts are necessary. Several shortest path algorithms are available for implementation. Even with the "restriction" of constant arc values, these algorithms are regularly implemented in areas of engineering, science, and business.

Dijkstra's algorithm for shortest path is considered one of the best and its intuitive framework is ideal as a base algorithm in development of this research effort [14].

Traditional optimization techniques have been criticized for their extensive assumptions. They yield mathematically pure solutions, however the stringent assumptions of the problem statements are often unrealistic. Therefore, the solution exists for a situation that may not exist. Simulation reduces the number of assumptions in the model and therefore offers the decision-maker more meaningful output. However, simulation also contains assumptions regarding input distributions.

Qualitative Discrete Event Simulation, abbreviated as QDES, is leading-edge research, which is well equipped to simulate models with interval-valued inputs. QDES combines Discrete Event Simulation, Event Graphs, and Qualitative Simulation. "Qualitative simulation is guaranteed to predict all real behaviors of systems consistent with the model" [5, p. 47]. This "quality" of Qualitative Simulation is essential in our model with interval values. The logical operators in interval mathematics, event graphs and discrete event simulation enable a QDES model to yield a complete output, which fully describes the model.

The QDES algorithm enumerates all possible events in the simulation and creates "threads" as outputs [7]. These threads represent all possible combinations of events. The secondary objective of this research is meaningful analysis of the qualitative shortest path network algorithm output. This research will attempt to consolidate the possibly large quantity of threads. Several techniques have been used in other areas in recent literature. This research will explore these techniques and attempt to offer a new alternative output analysis.

Networks

To appropriately review similar areas of research, some elementary graph theory notation and definitions are necessary. Definitions below are by Rockefeller [15].

- NETWORK: Two abstract sets N and A and a function that assigns each $j \in A$ to a pair (i, i') $\in \mathbf{NxN}$. The elements of N are called NODES and are represented pictorially by small circles. The cardinality of the node set N is conventionally given as n. The elements of A are ARCS, each arc is denoted by an ordered pair (i, j), where $i, j \in N$. Arcs are represented pictorially by arrows where the direction of the arrow shows the orientation of the arc. The cardinality of the arc set A is conventionally given as m.
- PATH: A path P in a Network G is a finite sequence $i_0, (i_0, i_1), i_1, \ldots, i_{r-1}, (i_{r-1}, i_r), i_r$ where each i_j is a node and (i_j, i_k) is an arc. The initial node of P is i_0 and the terminal node is i_r.
- CIRCUIT: A path in a network with the same initial node and terminal node.
- SIMPLE PATH: A simple (or elementary) path is one that uses no node more than once.
- POSITIVE PATH: A path containing only arcs a_{ij} traversed in the direction from node i to node j.
- CONNECTED: A network G is connected if for every pair of different nodes s and s', there is a path P: $s \rightarrow s'$. (P need not be a positive path.)
- ACYCLIC: A network G is acyclic if G possesses no positive circuits.

- $[S,S']^+ = \{j \in A \mid j \sim (i, i') \text{ with } i \in S, i' \in S'$, i.e. the set of all positive arcs from S to S';
- $[S,S']^- = \{j \in A \mid j \sim (i', i) \text{ with } i \in S, i' \in S'\}$, i.e. the set of all negative arcs from S to S'.
- COMPLEMENT: R\T is read "the complement of T in R"
- CUT: A cut in G is the signed arc set $Q = Q^+ \cup Q^-$ such that for some node set S, $Q^+ = [S, N\backslash S]^+$ and $Q^- = [S, N\backslash S]^-$ "The word "cut" for Q = [S, N/S] arises from the idea that any path P with initial node in S and terminal node in N/S must at some stage traverse one of the arcs in Q. A cut Q is shown in Figure 6.

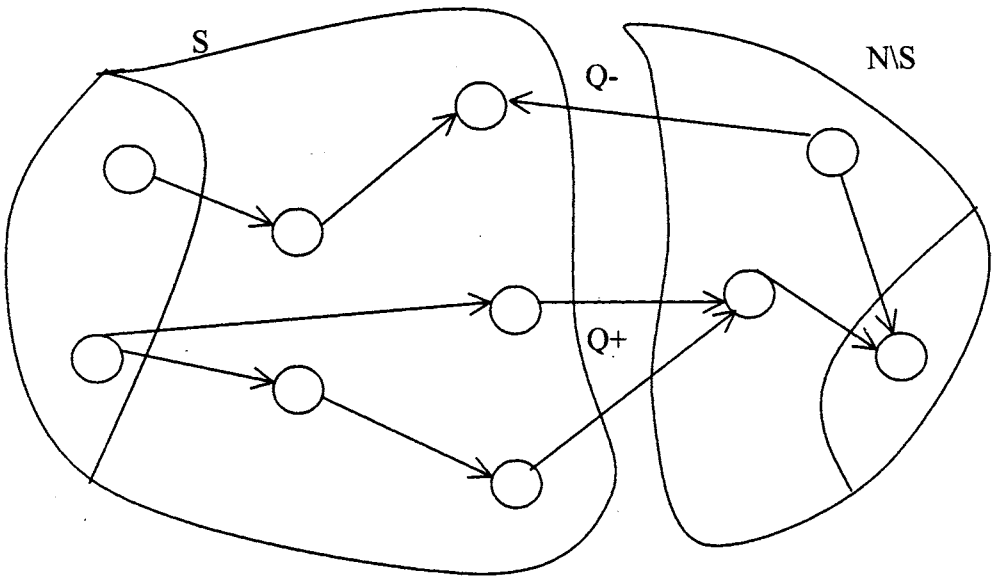


Figure 6: Cut [S,N\S]

The Shortest Path

Let G = (N, A, d) be a connected, acyclic network, where N is the set of nodes; A is the set of arcs, each element of A is of the form (i, j) for the arc connecting node i to node j; and c(i, j) is measure of performance along the arc, for every $i, j \in N$, $c(i, j) \geq 0$. Let N+ be the set of origin nodes. Let N- be the set of all destination nodes.

An arc can only be traversed directly from Node i to Node j if arc (i, j) exists, i.e. $c(i, j) < \infty$. A path moves from node to node by a specific arc connecting the two nodes. $\text{Node} \rightarrow \text{Arc} \rightarrow \text{Node} \rightarrow \cdots \rightarrow \text{Arc} \rightarrow \text{Node}$. Traversing each arc, (i, j), in the path, a measure of performance, e.g., cost of c(i, j) is incurred. The objective is to find a path from N+ to N- with the smallest measure of performance, e.g. smallest cost.

The following is a trivial example of a transportation network whose nodes are cities and arc lengths are traveling times between the cities. Figure 7 shows the network with nodes (cities) A, B, C, and D. The traveling times among these cities are shown on the arcs. Node A represents the location of a manufacturing facility and node D represents the location of the warehouse. The objective is to find the shortest traveling time from the manufacturing facility (Node A) to the warehouse (Node D).

One alternative to find the shortest path from Node A to Node D is to find all paths

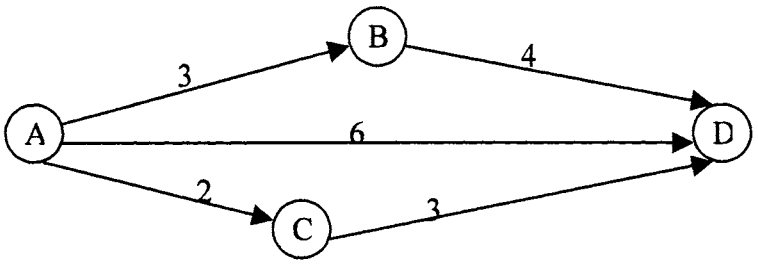


Figure 7: Transportation Network V: Nodes A-D

from A to D. Enumerating all possible paths from A to D yields:

Path 1 = Node A → Node B → Node D= 3 + 4 = 7 units

Path 2 = Node A → Node D = 6 units

Path 3 = Node A → Node C → Node D = 2 + 3 = 5 units

Therefore, Path 3 is the shortest path from A to D.

In amending this network by adding two nodes, A* and D*. These new nodes represent a new manufacturing city, A* and a new warehouse, D*. For this transportation problem, the objective is to find the shortest path from the set {A, A*} to the set {D, D*}. Figure 8 shows the modified network. If only transportation times are considered, where should the product be produced? Where should they be warehoused? What is the best shipping route?

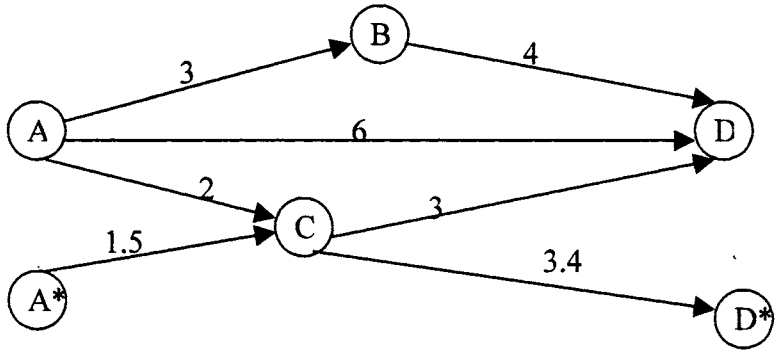


Figure 8: Transportation Network: Nodes A-D*

This network is only a slight modification to the first. However, it becomes obvious that enumerating all paths of a large network would be inefficient. More sophisticated techniques/algorithms have been established to solve this problem.

The predominantly used shortest path algorithms are Bellman-Ford's algorithm, Floyd-Warshall algorithm and Dijkstra's algorithm [14]. These algorithms use dynamic programming methods to exploit the fact that it is sometimes easiest to solve one

optimization problem by solving the problem for an entire family. These algorithms offer the decision maker bonus information. The dynamic programming structure of the algorithms yields not only the shortest distance from the origin node(s) to the destination node(s), but the shortest distances from the origin node to all other nodes. Additionally, the Floyd-Warshall algorithm offers the decision-maker the shortest distance between all nodes in the network.

Each algorithm has particular benefits and drawbacks. They offer a trade-off between information gained and computer time used. Each user has his/her opinion of the most useful shortest path algorithm.

The Bellman-Ford and Floyd-Warshall algorithms require only that the network contain no negative cycles (cycles whose cumulate length is negative). Dijkstra's algorithm can be performed on a graph with the condition that the graph contains no cycles and that all measure of performance (distances) along arcs are non-negative. Dijkstra's technique is the "most efficient option when given graphs satisfy further assumptions" [14, p. 440]. Dijkstra's technique is extremely intuitive and will be used as basis of the initial stage of this research.

Dijkstra's Method

Since Dijkstra's method for finding the shortest path in a network is a base for this research, the following section will be devoted to its presentation. "The essence of this procedure is that it fans out from the origin, successfully identifying the shortest path to each of the nodes of the network in the ascending order of their (shortest) distances from the origin, thereby solving the problem when the destination node is reached" [6, p. 411].

R. T. Rockefeller is a leading researcher in the area of network optimization. His book "Network Flows and Monotropic Optimization" covers a multitude of network optimization algorithms [15]. Below is Rockefeller's description of Dijkstra's shortest path algorithm. This algorithm introduces a positive and a negative direction on an arc. Although this research shall be restricted to only positive movement along an arc, this representation of Dijkstra's algorithm will include both positive and negative arc movements. Its addition adds only slight computational complexity. However, the algorithm below is a modification to Dijkstra's method in Rockefeller. The potential u_0 notation is omitted, due to the fact that our information is limited to only the minimum path and not a maximum tension problem.

Let

- G = (N, A, d) be connected, acyclic network, where N is the set of nodes;
- A is the set of arcs and each element of A is of the form (i,j) for the arc connecting node i to node j;
- $d^+(j)$, $j \sim (i, i`)$ is the length of the arc from node i to node i` (measure of traversing the arc in the positive direction); $d+(j) \geq 0 ; \forall\ j \in A$
- $d^-(j)$, $j \sim (i, i`)$ is the length of an arc from node i` to node i (measure of traversing the arc in the negative direction); $d-(j) \leq 0 ; \forall\ j \in A$
- N^+ be the set of origin nodes.
- N^- be the set of all destination nodes.
- $w : N \rightarrow \Re$; w is a function, which maps the nodes N to the reals, R. Let w(i) be the minimum distance from N+ to Node i $\forall k \in N^+$, $w(k) = 0$.

There are two sets S and T satisfying $N^+ \subseteq S \subseteq T \subseteq N \setminus N^-$. θ is a routing of T with base N^+, and w is defined $\forall$ T. Initially $T = S = N^+$, θ is empty, and $w \equiv 0$. To begin with, all nodes are "unscanned." Nodes will be *processed* one by one, and after a node is processed, the node will be said to have been "scanned."

Step 1. If there are no "unscanned" nodes in S (which is false initially) then go to step 3. Otherwise select any "unscanned" node i' which is an element of S and go to step 2.

Step 2. If there is an arc incident to i' and belonging to Q = [S, N\S] calculate

$$\gamma = \begin{cases} w(i`) + d^+(j) & \text{if } j \sim (i`,i) \\ w(i`) - d^-(j) & \text{if } j \sim (i,i`) \end{cases}$$ where i denotes the other node of j.

If $i \in T$ and $\gamma < w(i)$, redefine $\theta(i) = j$ and $w(i) = \gamma$.

If $i \notin T$ and $\gamma < \infty$, add i to T with $\theta(i) = j$ and $w(i) = \gamma$.

If $\gamma = \infty$ or $i \in T$ and $\gamma \geq w(i)$, do not change T, θ, or w.

Repeat this step for each arc j incident to i` and belonging to Q, then return to Step 1 (with i` henceforth regarded as "scanned")

Step 3. Calculate $\beta = \min\{ w(i) | i \in T\setminus S\}$

If $T\setminus S = \varnothing$, regard β as $+\infty$ and terminate;

Q is a cut of unlimited span (so [sup] = [min] = $+\infty$ in the two problems).

Otherwise add to S the nodes of T\S for which the minimum defining β is achieved. If node $i \in N^-$ is among these terminate; the θ-path $P: N^- \rightarrow i$ solves the min path problem.

Non-Constant Arc Distances

Due to the high demand for accurate network optimization, research is being performed in optimization of networks without the restriction of constant arc values. Chabini and Lan give a solution for "shortest path problems in dynamic networks, in which travel times are time dependent" [3, p. 60]. Chabini and Lan modify the A* algorithm first introduced by Hart, Nillson, and Raphael in 1968, which is specific to a origin-node to one destination-node problem variant. The A* algorithm attempts to be smarter than other shortest path algorithms. For example, consider a city network where the origin node is located at the center of the city and the destination node is in the far east. Other algorithms would typically put the same effort in searching to the east, west, north, and south of the origin node. "These algorithms may search through areas in through which the shortest path would not pass" [3, p. 63]. The A* algorithm chooses a set of nodes that have been reached and that are "candidates" for the selection of the next node.

Chabini and Lan use the A* algorithm to find the shortest path with cost $D = \{ d_{ij}(t) | (i, j) \in A \}$. The function d has an integer-valued domain and range and is therefore discrete and time-dependent. The work of Chabini and Lan is specific to the field of Intelligent Transportation Systems (ITS) and is motivated by the fact that "the computation of shortest paths is a fundamental component in route guidance systems and in the development of solution algorithms for the large-scale dynamic network flow models; such models are useful in supporting effective ITS decision-making" [3, p. 60].

The shortest path problem in this dynamic network is solved by applying a "static shortest path algorithm to its equivalent representation as time-expanded network. These

dynamic adaptations of the A* algorithm are based on effective lower bounds on minimum travel times that exploit the FIFO properties of dynamic data" [3, p. 73].

Chabini and Lan's work yields optimal solutions for many networks with extreme efficiency. The number of nodes searched and computer time is greatly reduced. However, the deterministic time-dependent arc values contain no option of variability. The proposed research will extend the non-deterministic shortest path problem to address the very real issue of variability in data, specifically arc measure.

Okada and Soper make the point that "As time or cost fluctuate with traffic conditions, payload and so on, it is not practical to represent each arc as a deterministic value" [13, p. 129]. Okada and Soper present work based on stochastic arc values. "... we are concerned with the shortest path problem of the network with each arc length represented as a positive fuzzy number, ...However, we cannot get an optimal solution in the normal sense because this type of problem is a so called "ill-posed" problem" [13, p. 129].

Okada and Soper solve the shortest path problem on a network with fuzzy arc lengths. A fuzzy number $\tilde{a}$ is an upper semi-continuous, normal and convex fuzzy subset on the real line $\Re$ such that $\mu_a : \Re \rightarrow [0,1]$ where μ_a is the membership function of a. "A flat fuzzy number $\tilde{a}$ is a fuzzy number such that $\exists$ $(\underline{m}, \bar{m}) \in \Re, \underline{m} < \bar{m}, \text{and} \mu_a(x) = 1 \ \forall x \in [\underline{m}, \bar{m}]$" [13, p. 130]. An L-R type flat fuzzy number is denoted as $(\underline{m}, \bar{m}, \alpha, \beta)_{LR}$, where α, β are left-hand and right-hand spreads, as defined as follows:

$$\mu_a(x) = \begin{cases} L(\underline{m} - x)/\alpha & \text{if } x < \underline{m}, \alpha \in \Re^+ \\ 1 & \\ R(x - \overline{m})/\beta & \text{if } x > \overline{m}, \beta \in \Re^+ \end{cases}$$

where L and R are even functions such that L(0) = R(0) = 1 and L(1) = R(1) = 0, and L and R are strictly decreasing on $(0, \infty)$.

Addition of flat fuzzy numbers, given by the symbol, $\oplus$, is defined as $(\underline{a}, \overline{a}, \alpha, \beta) \oplus (\underline{b}, \overline{b}, \gamma, \delta) = (\underline{a} + \underline{b}, \overline{a} + \overline{b}, \alpha + \gamma, \beta + \delta)$. The characteristics of the $\oplus$ function for flat fuzzy numbers enables the enumeration of the cumulative distance along a path. One method to rank fuzzy numbers is "mapping each fuzzy number to the real line, where total order exists" [13, p. 131]. The other method of ranking is that "the decision maker a priori chooses a degree of conformity for which the inequality may be considered true" [13, p. 131]. That is, the comparison between two fuzzy numbers is a fuzzy operation defined by the decision-maker. There exist several different theoretical approaches for comparing fuzzy numbers [13].

Additionally, there exist many different types of fuzzy numbers. Okada and Soper have chosen to use trapezoidal fuzzy numbers to represent each arc length. "Each arc length (duration time) in the generated network is converted into a trapezoidal fuzzy number with a flat part of about 20% and spreads about 10% of the original duration time" [13, p. 137].

The shortest path is formulated by the following linear program:

$$\min \tilde{f}(x) = \sum_{(i,j) \in A}^{\oplus} \ell_{ij} x_{ij}$$

$$\text{s.t.} \quad \overset{\oplus}{\sum_j} x_{ij} - \overset{\oplus}{\sum_j} x_{ji} = \begin{cases} 1 & \text{if } i = s \\ 0 & \text{if } i \neq s,t \left(i = 1,\ldots,n\right) \\ -1 & \text{if } i = t \end{cases}$$

$x_{ij} = 0$ or 1 for any $(i,j) \in A$.

Where $\overset{\oplus}{\sum}$ in the objective function means the addition $\oplus$ between fuzzy numbers.

However, "It impractical to solve even a small size problem by using the network simplex method due to the increase in decision variables" [13, p. 134]. Okada and Soper eventually use a generalization of Dijkstra's algorithm to solve the simple shortest path problem.

Okada and Soper's solution to the shortest path problem is most interesting. The fuzzy arithmetic allows for a mathematical solution to this stochastic process. Due to the fact that path lengths are not real numbers, all paths are enumerated and the number of obtained paths cannot be controlled by the decision maker. Okada and Soper have analyzed the multi-path output by attempting to find non-dominating paths and possible shared sub-paths. This research effort also uses the uncertainty measures of non-dominance and shared sub-paths in the algorithm output methodology. Although Okada and Soper's model contains minimal assumptions, the fuzzy arc lengths are more limiting than interval valued arcs.

Sudharasan has used fuzzy distances in a routing algorithm analysis optimal web path estimation. "A routing algorithm sets up paths to connect the different nodes in the network and strives to optimize performance measures like distance, mean packet delay and network throughput" [18, p. 380]. Sudharasan notes the inherent uncertainty in this dynamic information because it is at least as old as the distance between nodes, as the

changes in the status of the network are not immediately known to every node in the network. "Traditional network routing algorithms have not attempted to deal with this uncertainty in any way and most behave as though there is no uncertainty in the information available at a node" [18, p. 380].

Network optimization is not limited to finding the shortest path. Much work has been done in finding a minimal spanning tree in a network as well. A graph S is a minimum spanning tree of G = (N, A, d) if S is a connected graph with no elementary circuits, such that S = (N*, A*, d) where N*=N and $A^* \subseteq A$, n(A*) = n(N*)-1. Finding solutions to the minimal spanning tree problem with non-constant values is a great benefit in many industrial applications, telecommunications networks, and rail transportation networks.

Yaman, Karasan, and Pinar analyze the robust spanning tree problem with interval data. "The combination of the interval uncertainty with robustness is attractive ... we do not have to specify a distribution for the data, nor its moments, which is not always easy" [20, p. 31]. No probability distribution is assumed for edge costs. Yaman et al. finds the lower and upper cost of a spanning tree; the cost is calculated by the sum of the arcs in the tree at their lower and upper value respectively. Yaman et al. uses the bound values only in his algorithm to hedge against the worst possible delay in a robust spanning tree.

Chen and Lin use interval arc values in determining the optimum location of the 1-median of a tree. A 1-median is a node in the tree such that the distance from that node to all other nodes is minimized. This is the network model of the placement of a distribution center among many locations. Each location is a node and the distances between locations are arc lengths. Finding the 1-median of this network will yield the optimum placement for the distribution center.

Chen and Lin reiterate the importance of interval-valued arc lengths. "In some practical environments, it is sometimes difficult to characterize the uncertainty involved in input data by probabilistic distributions with reasonable accuracy, especially if limited past data are available" [4, p. 94]. Chen and Lin find the set of all 1-median nodes for some feasible scenario. Chen and Lin replace edge-length intervals with their upper bounds to find the worst case scenario. A 1-median is chosen to minimize the maximum regret against this worst case. The generation of all feasible scenarios is not discussed due to the fact that the analysis is based primarily on a robust approach to solving the problem. "The robust approach, which aims at a decision with minmax regret is more suitable when limited information is available about the uncertainty involved" [4, p. 94]. The 1-median is chosen as node x such that the difference between the spanning length for node x and the spanning length for node y in scenario s is minimized for all nodes s.

The optimization of networks with non-deterministic arc values has proven itself as a worthy research topic. Few attempts have been made to find optimal solutions for these networks. Specifically, research with regard to networks with interval arc values has been analyzed from a min-max approach. Research has yet to solve the problem of finding optimal path(s) for the shortest path in a network.

To appropriately analyze the optimal shortest paths, it is necessary to enumerate all shortest paths for a network. Techniques of Qualitative Discrete Event Simulation are an ideal tool for the formulation of all possible shortest paths of a network. QDES is a combination of techniques of discrete event simulation, event graphs, and qualitative simulation.

Simulation

Kelton, Sadowski, and Sadowski provide a fundamental definition of simulation. "Simulation refers to a broad collection of methods and applications to mimic the behavior of real systems, usually on a computer with appropriate software" [8, p. 3]. The software available for simulation is extensive due to the popularity of this operations research tool. The early implementation of simulation software was through programming languages developed specifically for the task. GPSS (1961), SIMSCRIPT (1962), GASP (1974), SLAM (1986), SIMAN (1995) and ProModel (1994) are several successfully implemented programming languages [8, 9].

"Over the last two decades or so, simulation has been consistently reported as the most popular operations research tool" [8, p. 7]. Manufacturing plants, personal-service operations (e.g. banks), distribution networks, computer networks, supermarkets, theme parks, freeway systems, and emergency response systems can be modeled in simulations. This list is certainly not intended to be exhaustive, as the quantity of simulation models is extensive. "The main reason for simulation's popularity is its ability to deal with very complicated models with correspondingly complicated systems. This makes it a versatile and powerful tool" [8, p. 8].

A specific criticism of simulation is the formulation of and output resulting from random inputs. The output of a simulation will vary considerably depending on the distribution from which input variables are drawn. Because many real systems are affected by uncontrollable and random inputs, many simulation models involve random, or stochastic, input components, causing their output to be random too.

Law and Kelton categorize simulations as deterministic vs. stochastic, static vs. dynamic, and continuous vs. discrete. In addition, there is the distinction between quantitative and qualitative simulation as well. Below are the definitions of quantitative and qualitative according to the American Heritage Dictionary of The English Language: [10].

- Quantitative: Expressed or capable of expression as a quantity.
- Qualitative: Of, pertaining to, or concerning quality or qualities.

Generally speaking, systems modeled using quantitative simulations contain input variables that take on one specific quantity during the simulation. "Quantitative models include discrete event simulation, min-max algebra, Markov chains, stochastic Petri nets, queues, and queueing networks" [19, p. 2]. These quantities are often stochastic (i.e. are generated from some probabilistic distribution) in discrete-event simulation models.

However, systems modeled using qualitative simulations contain input variables that may take on several quantities. "Qualitative Models capture logical aspects of system evolution" [19, p. 2]. For the work proposed here, the variables have an infinite cardinality and may take any value within a specified interval. Logical choices are made throughout the simulation to create the simulation output. "An important aspect of qualitative simulation is that the behavioral prediction can branch, corresponding to qualitatively distinct futures that cannot be discriminated by the available information. Qualitative simulation is guaranteed to predict all real behaviors of systems consistent with the model" [5, p. 47].

Fouche and Kuipers give an overview of qualitative simulation. Qualitative simulation is based on the observations that:

- The domain of a variable representing a physical parameter of a system can often be partitioned into a small number of "landmark" points and intervals between them, which represent real qualitative distinctions for the magnitude of the variable;
- Knowing the direction of the change of a variable, in conjunction with its qualitative magnitude, is often enough to determine the qualitative properties of its evolution; and
- For determining the qualitative behavior of a system, it is often adequate to know a functional relationship between two variables down to monotonicity and corresponding pairs of landmark values. [5, p. 47].

At the 1991 Winter Simulation Conference, Cellier chaired "Qualitative Modeling and Simulation: Promise or Illusion", a panel discussion to discuss the practical applications of qualitative simulation. In the Proceedings article, Cellier defines qualitative variables as either nominal measures, ordinal measures, interval measures or ratio measures. Where nominal measures are variables with exhaustive and mutually exclusive characteristics; ordinal measures are variables that are nominal and rank ordered; interval measures are variables that are ordinal and in which any two interval measures can be added to or subtracted from one another; and ratio measures are variables that are interval and contain a true zero point. Cellier categorized qualitative models into four types: naive physics models, inductive reasoning models, symbolic discrete-event models, and neural models. In Cellier's discussion of symbolic discrete-event models he states: "Symbolic discrete-event simulation generates all trajectories that are feasible due to the fuzziness of these parameters" [2, p. 1089]. This is a truly unique attribute of QDES.

Another distinction between qualitative discrete event simulation and traditional DES is the manner in which the simulation evolves. "Discrete-event simulation concerns the modeling of a system as it evolves over time by a representation in which state variable change instantaneously at separate points in time" [9, p. 7]. The emphasis of

SIMSCRIPT (1962), GASP (1974), SLAM (1986), SIMAN (1995) and ProModel (1994) simulation languages is modeling the process of an entity traveling through the model.

However, qualitative discrete-event simulation (QDES) has an alternative, event approach to formulating models.. It is the events themselves that dominate and control the simulation. Schruben first introduced the concept of modeling simulations by event graphs in 1983. An event graph is a collection of that are associated with other events through the model structure [7].

Simulation and Event Graphs

Schruben and Yucansan formalized Simulation Graphs as an extension of Event Graphs during the 1988 Winter Simulation Conference. Figure 9 shows an edge that is a part of a simulation graph. "Pictorially the vertices of an event graph represent state

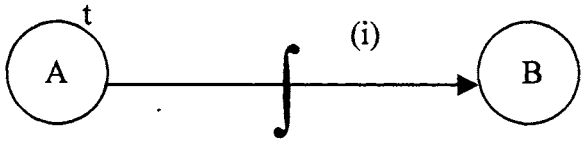


Figure 9: Scheduling Edge

changes that are associated with the various events in the simulation. The edges of a graph represent the logical and temporal relationships between the vertices" [17, p. 504].

The edge is interpreted as follows: whenever event A occurs, if condition (i) is true then event B will be scheduled to occur t time units later. "The elements of a simulation model are the state variables, events that change the values of state variables, and the relationships between the events. An event graph is a structure of the objects in a discrete-event system that facilitates the development of a correct simulation model" [17,

p. 504]. Basically, the edges define the necessary conditions and time delay between one event and another. In the formalization of simulation graphs and models, Schruben and Yucansan defines a graph as "an ordered quadruple G = (V(G), $E_S(G)$, $E_C(G)$, $\Psi(G)$) where V(G) is the vertex set of G, $E_S(G)$ is the set of scheduling edges, $E_C(G)$ is the set of canceling edges, and $\Psi(G)$ is the incidence function" [17, p. 505]. This definition is nearly identical to a traditional network with the addition of the canceling edge, $E_C(G)$. Graphically, the canceling edge and scheduling edge are identical. In the graphical definition, they are distinguished as two distinct sets and their operations are certainly not identical. The description of the simulation model is more extensive. Schruben and Yucasan define a simulation model as a structure of indexed sets.

- $\Im = \{f_i : STATES \rightarrow OUTPUTS | \ i \in V(G)\}$, which is the set of state transitions associated with vertex (event) i.
- $\zeta = \{c_{ij} : STATES \rightarrow \{0, 1\} | \ (i, j) \in E_S(G) \cup E_C(G)\}$, which is the set of edge conditions;
- $T = \{t_{ij} : (i, j) \in E_S(G) \cup E_C(G)\}$,which is the set of edge delay times;
- $\Gamma = \{v_i : i \in V(G)\}$,which is the set of event execution priorities; note that Γ is the set of nonnegative integers; assuming that smaller integers will correspond to higher priorities with 0 representing the highest execution priority;
- S_i is the set of state variables possibly altered by event vertex i, $i \in V(G)$;
- E_i is the set of state variables involved in the conditions on the arcs emanating from vertex i, $i \in V(G)$;
- Z is the list of scheduled events (events list)
- T is the global simulation clock

The simulation also includes a stochastic process of independent uniform random variables on (0, 1).

Schruben and Yucasan note that "A graph G can be thought of as a triple, G = ((V(G), E(G), Ψ (G))" [17, p. 506]. This definition of a graph contains only one type of event conditions. In the proceedings of the 1995 Winter Simulation Conference, Savage and Schruben presented "Eliminating Event Cancellation in Discrete Event Simulation" [16]. Savage and Schruben describe the procedure of the elimination of canceling edges.

> Let v be the event that may be canceled, v_{ch} be the added check event and v_x represent an event that might cancel event v. The following steps replace the canceling edge:
>
> 1) Remove the canceling edge (v_x, v).
> 2) Create new state variables VC (to count the cancellations of v) and SV (= 1 if v has not been canceled, 0 otherwise).
> 3) Add {VC = VC +1} to the state changes of v_x.
> 4) Add the vertex v_{ch} with state change: {SV=1 iff (VC = 0), VC = max(VC-1, 0)}.
> 5) Add the edge (v_{ch},v) with edge condition c_e = SV
> 6) Replace all scheduling edges (v^t, v_{ch}) with the same time delay and edge conditions. [16, p. 747].

Ingalls analyzed the difference in calendar management (i.e. scheduling and canceling events) and the implementation of edge execution conditions. "The addition of the edge execution conditions to the simulation graph methodology provides a standard, complete methodology for modeling interruption. Edge execution conditions provide a simpler and more efficient management of these calendars" [7, p. 52].

Temporal Intervals in Qualitative Simulation Graphs

In QDES, simulation graphs are implemented with conditions formulated by Schruben (1988) with the additional characteristic of interval-valued state variables. Ingalls emphasizes that "The purpose for describing state variables with interval values is

to allow the user to describe the inherit uncertainty of the decision maker or modeling with it comes to the true value of the variable" [7, p. 24].

Table I
Allen's Interval Algebra for Intervals $t = [t^-, t^+]$ and $s = [s^-, s^+]$

Relation	Symbol	Symbol for Inverse	Definition	Example
t before s	<	>	$t^+ < s^-$	TTT SSS
t equals s	=		$(t^- = s^-)$ and $(t^+ = s^+)$	TTT SSS
t overlaps s	o	oi	$(t^- < s^-)$ and $(t^+ > s^+)$ and $(t^+ < s^-)$	TTT SSS
T meets s	m	mi	$(t^+ = s^-)$	TTTSSS
t during s	d	di	$((t^- > s^-)$ and $(t^+ \leq s^+))$ or $((t^- \geq s^-)$ and $(t^+ < s^+))$	TTT SSSSS
T starts s	s	si	$(t^- = s^-)$	TTT SSSSSS
t finishes s	f	fi	$(t^+ = s^+)$	TTT SSSSSS

The implementation of interval valued state variables in a simulation graph can be mathematically rigorous. In 1983, Allen developed an algebraic system of time intervals [1]. Ingalls supplements the algebra by adding several operators. Let $t = [t^-, t^+]$ and $s = [s^-, s^+]$ be closed intervals on $\Re$. Tables I and II show the relations developed by Allen and Ingalls, respectively. Additionally, Ingalls implemented logical extension such as (&), or (|) and negation (!). These algebraic operations enable a QDES model to appropriately analyze each next event in the simulation. When the order of events is uncertain, a QDES simulation will try all the combinations of the event and create a thread for each event in the set. Each thread has its own future events calendar for that thread. Each thread maintains its own calendar and calendar time [7].

Table II

Ingalls' Interval Algebra for Intervals $t = [t^-, t^+]$ and $s = [s^-, s^+]$

Relation	Symbol	Symbol for Inverse	Definition	Example
t intersects s	i		$\max(t^-, s^-) \leq \min(t^+, s^+)$	TTT SSS
t + s	+		$[t^- + s^-, t^+ + s^+]$	
t - s	-		$[t^- - s^-, t^+ - s^+]$	
inverse(t)	inverse		$[1/t^+, 1/t^-]$	
t*s	*		$[t^- * s^-, t^+ * s^+]$ if $t^- * s^- \leq t^+ * s^+$	
t/s			t* inverse(s)	
combine(t,s)	combine		$[\min(t^-, s^-), \max(t^+, s^+)]$ if (t i:s)	
midpoint(t)			$(t^- + t^+)/2$	
width(t)			$t^+ - t^-$	
max(t,s)	max		$[\max(t^-, s^-), \max(t^+, s^+)]$	
min(t,s)	min		$[\min(t^-, s^-), \min(t^+, s^+)]$	
intersection(t,s)	$\cap$		$[\max(t^-, s^-), \min(t^+, s^+)]$, if $(\max(t^-, s^-) \leq \min(t^+, s^+))$ $\varnothing$ otherwise	

Ingalls extends the general simulation graph framework introduced by Schruben to accomplish this extensive task. Ingalls developed a QDES algorithm with interval state variables. The algorithm was implemented with the analysis of a standard inbound-processing-outbound model.

As QDES output consists of all possible threads (i.e. outcomes), the number of threads explodes as the simulation progresses. Therefore, QDES is most beneficial with problems containing a fixed time horizon. A fixed horizon problem would analyze a graph for a short period of time, or possibly a graph that is finite in nature. The shortest path of a network is finite in nature if the graph contains a finite number of nodes. Ingalls developed a QDES algorithm for PERT scheduling with and without resource restrictions. Ingalls converts the network into an Event Graph. After implementing the QDES algorithm for PERT analysis, the resulting thread output is extensive. Ingalls uses Thread Scoring Methods to analyze the numerous threads. Ingalls asserts that minimal

assumptions need to be made about the intervals in order to score them. "One of our methods is more qualitative, in that we rank the intervals by the midpoint of the interval. Another uses the earliest midpoint as the basis and determines a relative score. The third method introduces a minimal statistical assumption, namely that the interval is uniformly distributed" [7, p. 125].

In Ingalls' comments regarding future research pertaining to QDES, he states "One of the most interesting future research topics coming out of this dissertation is the possibility of exact output statistics from the simulation" [7, p. 130]. It is a fundamental objective of this research to develop a cohesive analysis of the output resulting in the QDES algorithm specific to the shortest path problem. The output methodology consists of techniques of min-max regret, non-dominating paths, and critical sub-paths.

The ability to analyze a model with minimal assumptions is well established as a meaningful research objective. Several researchers have published literature yielding solutions for several network optimizations with non-constant arc values. However, a solution to the shortest path of a network with interval-valued measures of performance does not exist. This research effort has developed such an algorithm yielding the shortest paths of this network and its accompanying output analysis.

CHAPTER III

THE ALGORITHM

Algorithm Overview

The primary objective of this research effort was the formulation of an algorithm which would generate the set of all shortest paths from Node 1 to Node n (End Node) in an acyclic, directed network with ordered node set N, arc set A, and interval-valued set c, the measure of performance for traversing arcs. An algorithm was developed which met this objective. It was coded in Visual Basic [11] with a Microsoft Excel [10] interface. The Visual Basic Code is in Appendix A.

The mathematical form of this statement is given in the following definition.

Given

$G = (N, A, c)$, i.e., G is acyclic and directed

$N = \{1, 2, ..., n\}$

$A = \{(a, b) \mid (a, b) \in A \rightarrow a < b\}$

$c(a, b) = [\text{lower bound, upper bound}] = [L_{ab}, U_{ab}]$ and c is the measure of performance of arc.

Find

$$\{P \mid P(1, ..., a, b, ..., n) \text{ and } C = \min \sum_{(a,b)\in P} c^*(a, b) \text{ for } c^*(a, b) \subseteq c(a, b)\}.$$

Note that G is an acyclic, directed network N with n nodes from 1 to n. The node set N is an ordered set such that if there exists an arc from Node a to Node b then $a < b$. Dijkstra's general shortest path method finds the shortest path(s) of an undirected network G from a set N- to the set N+. This research effort contains a different set of

assumptions than Dijkstra's general shortest path method. Specifically, Dijkstra's algorithm does not contain the acyclic assumption and the single initial node and single terminal node assumption. However, finding the shortest path(s) of a directed network with single nodes for N- (Node 1) and N+ (Node n) involves no loss of generality [13]. The fundamental difference between network G and the general shortest path network is the structure of the measure of performance along an arc. The measure of performance along an arc is interval-valued; that is, the measure of performance can be any number greater than or equal to the lower bound and less than or equal to the upper bound of the interval. This specific type of network shall be referred to as an "interval-network".

The interval nature of the measure of performance in G adds significant complication to the algorithm, which generates the shortest path(s). In an interval-network, the cardinality of the shortest path set may be extremely large. Although a general network may have multiple shortest paths, the occurrence of multiple shortest paths does not complicate the generation of these paths. Figure 10 shows a network with constant-valued measures of performance with more than one shortest path. The network in

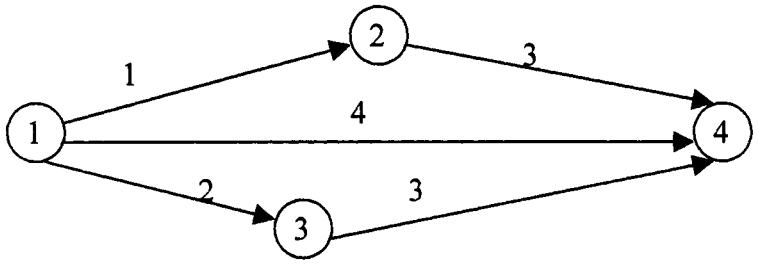


Figure 10: Transportation Network I: Nodes 1-4

Figure 10 has two shortest paths from Node 1 to Node 4. For convenience of presentation, the cumulative measure of performance along a path will be referred to as

the path length, although it may correspond to any measure defined by the user. Both paths have a length of 4 units.

Path 1 = Node 1 → Node 2 → Node 4

Path 2 = Node 1 → Node 4.

In an interval-network, multiple paths may exist for a specific path length, similar to the path of length 4 units in the general network in Figure 9. Additionally, there are numerous possibilities for the length of the shortest path depending on the measures of performance of the arcs. The shortest paths are generated by analyzing the subsets of measures of performance along the arcs. A shortest path set is generated for each specific subset of the measure of performance along an arc. The subsets are created by logic describing all possible minimum values. These are "cut-points" of the measure of performance interval. The cut-points are found in sub-procedure Identify_Cut_Points and they are ordered in sub-procedure Order_Cut_Points. The algorithms for these sub-procedures are provided in Appendix B and Appendix C, respectively.

In Dijkstra's method for finding shortest path of a traditional network, w(k) is defined as the shortest distance from N^+ to Node k. In the interval network, w(k) will have the same definition. As in a general network, w(k) in the interval-network is fundamental in the development of the shortest path. However, the value of w(k) is dependent on the subsets of measures of performance. Several w(k) values may exist for every node k. The set of $w_i(k)$ will be defined as the set of node threads.

An amended network is created based on each thread for w(k). The network is modified to include only arcs and arc values that can be contained in a shortest path. All shortest paths are generated from this amended network in the sub-procedure Find_Paths.

Because of the complicated nature of the interval-network, the algorithm generates shortest paths with arc values that may appear counter-intuitive. That is, the lower bound is greater than the upper bound. The sub-procedure Change_Arc_On_Path changes specific arc values to form a more meaningful solution set.

The main procedure of the algorithm first finds and orders the cut-points of the nodes beginning at Node 1 and ending at Node n. The network is then amended by replacing nodes with corresponding node*threads and by specific arcs connecting the node*threads. All possible paths are then found. Subsequently, the measures of performance corresponding to specific arcs along the paths are changed where necessary.

The flowchart of the main procedure Avery_Shortest_Path_Algorithm is shown in Figure 11. The remainder of this chapter will give a more complete description of the following sub-procedures : Generate_w(k), Check_Arcs, Find_Paths, and Change_Arc_On_Path. The Change_Arc_On_Path sub-procedure is supplementary to the main algorithm, as the measure of performance along the arc is not a primary outcome to the shortest path algorithm. The algorithm generates all possible shortest path distances and all shortest paths of the network that have the opportunity of having a specific shortest path distance. Due to the uncertainty of a qualitative network, the shortest path solution set consists of the shortest paths which traverses the amended network's node*threads. The main algorithm description of the generation of the shortest paths is given below.

Algorithm Steps

Given
G = (N, A, c), i.e., G is acyclic and directed
N = {1, 2,..., n}
Node 1 is the originating node
Node n is the terminating node
$A = \{(a, b) \mid (a, b) \in A \rightarrow a < b\}$

c(a, b) = [lower bound, upper bound] = $[L_{ab}, U_{ab}]$ and c is the measure of performance of arc.

Find

$$\{P \mid P(1,..., a, b,..., n) \text{ and } C = \min \sum_{(a,b)\in P} c^*(a, b) \text{ for } c^*(a, b) \subseteq c(a, b)\}.$$

Step 0: INITIALIZE

$w_1(1) = [\hat{L}_{11}, \hat{U}_{11}] = [0, 0]$
CardW (Node 1) = 1
$w_k(a) = [\hat{L}_{ak}, \hat{U}_{ak}]$ = empty,
CutPoints(Node a, Position) = empty,
OrderedCutPoints(Node a, Position) = empty,
Stopping Cut Point(Node a) = 9999,
CardW(Node a) = empty,
ArcValid(Originating Node*Thread, Terminating Node*Thread) = false.
Path Node*Thread (Path Number, Position, Length) = empty
NOP(Node*Thread) = NOP(a*j) + 1 (NOP = Number of Paths)
LPP(Path Number, Node*Thread) = empty (LPP = Last Path Position)

MAIN ALGORITHM
(For each node beginning with Node 1, Iterate from Step 1 to Step 6)

For Node b = 2 to n
(Step 1 – Step 6)

For a = 1 to b-1, (a = predecessor node)
(Step 1 – Step 2)
Index = 0

Step 1: "Find Unordered CutPoint Set A"
d = CardW(a)
Recall: $w_p(a) = [\hat{L}_{ap}, \hat{U}_{ap}]$, $c(a, b) = [L_{ab}, U_{ab}]$
Index = Index + 1
CutPoints(Node b, Index) = $\hat{L}_{ap} + L_{ab}$,
Index = Index + 1
CutPoints(Node b, Index) = $\hat{U}_{ap} + U_{ab}$

Step 2: "Find Unordered CutPoint Set B and Possible Stop CutPoint Set from Predecessor Nodes"
Index = Index + 1
CutPoints(Node b, Index) = $\hat{L}_{ap} + L_{ab}$,
Index = Index + 1
CutPoints(Node b, Index) = $\hat{U}_{ap} + U_{ab}$

If ($\hat{U}_{ap} + U_{ab}$) < Stopping Cut Points(b)

Then Stopping Cut Points(b) = $\hat{U}_{ap} + U_{ab}$

Next a

Step 3: "Create Ordered CutPoints"

From CutPoints(Node b, Index) create the set OrderedCutPoints such that:

For every o,

OrderedCutPoints(Node b, Position o) < OrderedCutPoints(Node b, Position o+1)

Step 4: "Find Cardinality of Node Set W(b)

CardW(b) = position of StoppingCutPoint(b) in OrderedCutPoints Set-1.

Step 5: "Find Node Set W(b)"

For k = 1 to CardW(b)

$\hat{L}_{bk}$ = OrderedCutPoints(Node b, Position k)

$\hat{U}_{bk}$ = OrderedCutPoints(Node b, Position k+1)

$w_k(b) = [\hat{L}_{bk}, \hat{U}_{bk}]$,

Next k

Next Node b

Go to Step 1

Step 6: "Find Possible/Valid Arcs on Shortest Path"

Recall, $w_k(b) = [\hat{L}_{bk}, \hat{U}_{bk}]$, $w_j(a) = [\hat{L}_{aj}, \hat{U}_{aj}]$,

For b = 1 to n-1

For k = CardW(b)

For a = b+1 to n

For j = 1 to CardW(a)

If $\hat{L}_{bk} - \hat{L}_{aj} \geq \hat{L}_{ab}$ and $\hat{U}_{bk} - \hat{U}_{aj} \leq \hat{U}_{ab}$ Then ArcValid (b*k, a*j) = True.

Next j

Next a

Next k

Next b

* The Paths generated by the main algorithm are all paths of the amended network consisting of node*threads and valid shortest path arcs.

Step 7: "Find all Shortest Paths from Possible Arcs on Shortest Path"

For a = 2 To n

For j = 1 To CardW(a)

If ArcValid(1*1, a*j) = True Then

Path Node*Thread(1, 1, a*j) = 1*1

Path Node*Thread(1, 2, a*j) = a*j

NOP(a*j) = NOP(a*j) + 1

```
        LPP(1, a*j) = 2

For b = 2 To a - 1
For k = 1 To CardW(b)
If ArcValid(b*k, a*j) = True Then
For Path Index = 1 To NOP(b*k)
        For Position Index = 1 To LPP(Path Index, b*k)
                Path Node*Thread(NOP(a*j) + Path Index, Position Index, a*j) =
                        Path Node*Thread(Path Index, j, b*k)
        Next Position Index

        Path Node*Thread(NOP(a*j) + Path Index, LPP(Path Index, b*k) + 1, a*j) = a*j
        LPP(NOP(a*j) + Path Index, a*j) = LPP(Path Index, b*k) + 1
Next Path Index

NOP(a*j) = NOP(a*j) + NOP(b*k)

Next k
Next b
Next j
Next a
```

Generate w(k)

The Avery Shortest Path Algorithm, which finds the shortest path(s) of an interval network, has an identical structure to Dijkstra's method for a general network. That is, it fans out from the origin, identifying the shortest distance from the originating node to each of the nodes of the network in the ascending order of their distances from the origin. The w(k) values are the shortest distances from Node 1 to Node k. These w(k) values are the essence of the algorithm. In Dijkstra's method for finding the shortest path of a general network, w is a function, which maps the nodes to the real numbers. For interval -networks, the measure of performance along an arc is interval-valued. Therefore in an interval-network, w is a relation between the nodes and interval values. That is, the values of w(k) are intervals.

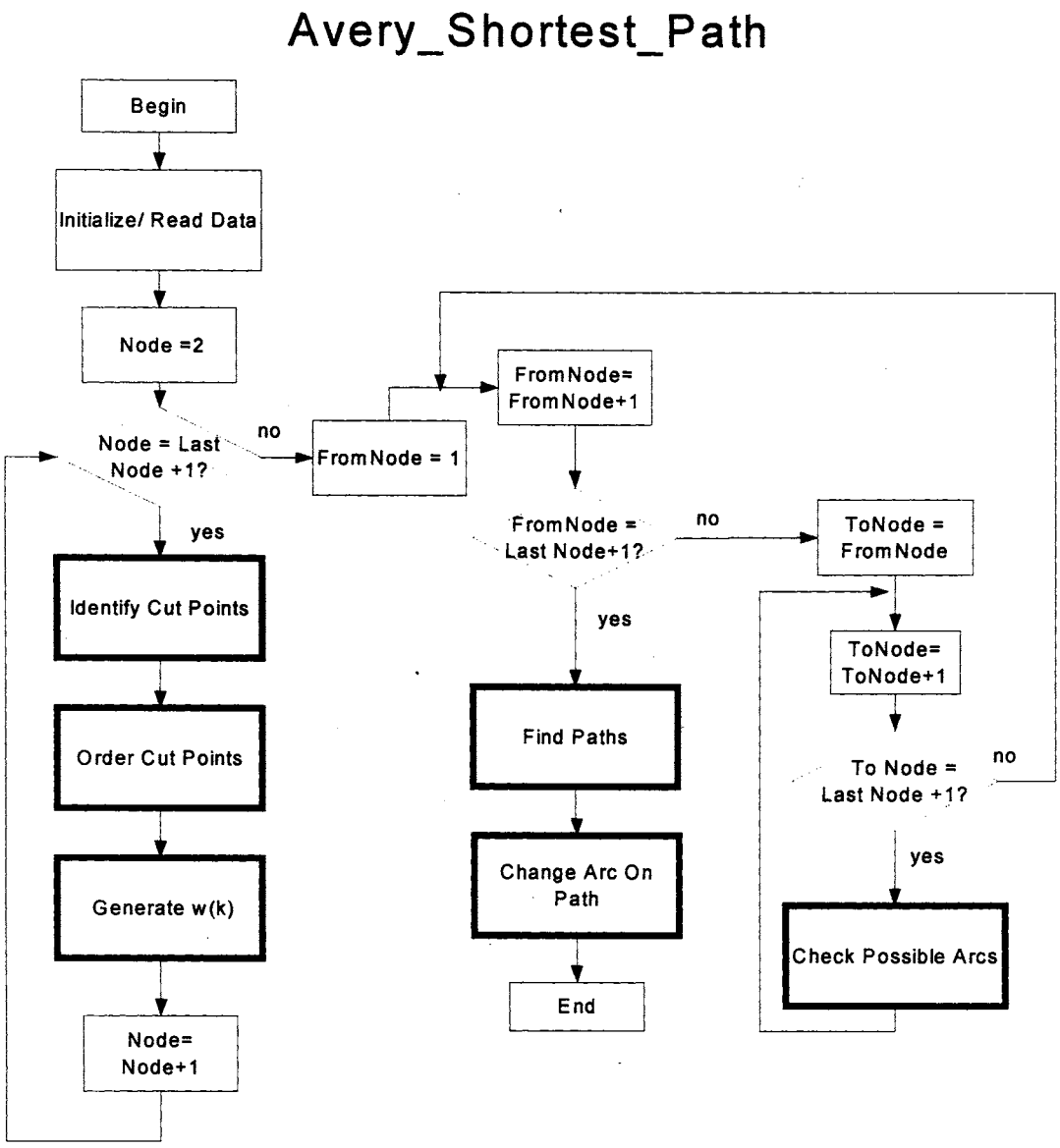


Figure 11: Flow Chart Avery_Shortest_Path

Let Omega(j, k) = w(j) + c(j, k). Recall that c(j, k) is the interval-valued measure of performance of the arc connecting Node j and Node k. Omega values can be calculated using basic principles of interval math, i.e. [a, b] + [c, d] = [a + c, b + d] [1]. Omega(j, k) represents the shortest path length from Node 1 to Node k that traverses Node j immediately before traversing Node k.

$$w(k) = \begin{cases} \min\{\text{Omega}(j, k)\}\ \forall \text{arc}(j, k) \text{ where Node } j \text{ preceeds Node } k & k = 2,\ldots,n \\ [0, 0] & k = 1 \end{cases}$$

In an interval-network, a unique value for w(k) is not guaranteed. The interval-valued minimum is the interval in which the true minimum lies. w(k) may (and often will) consist of several values. Actually, the minimum will depend directly on the measure of performance of predecessor arcs. The minimum is found on a case-by-case basis.

The $w(k) = [L_k, U_k]$ for the network shown in Figure 12 can be calculated without complication. Since, only Node 1 precedes Nodes 2 and 3

$w(2) = w(1) + c(1,2) = [0, 0] + [1, 2] = [0 + 1, 0 + 2] = [1, 2]$

$w(3) = w(1) + c(1, 3) = [0, 0] + [2, 4] = [0 + 2, 0 + 4] = [2, 4]$

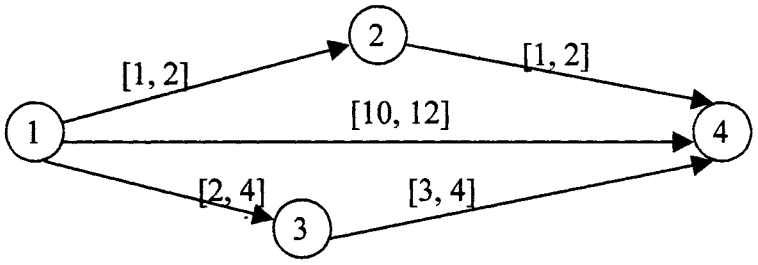


Figure 12: Transportation Network II: Nodes 1-4

The Nodes that precede Node 4 are Nodes 1, 2 and 3.

$\text{Omega}(1, 4) = w(1) + c(1, 4) = [0, 0] + [10, 12] = [10, 12]$

$\text{Omega}(2, 4) = w(2) + c(2, 4) = [1, 2] + [1, 2] = [2, 4]$

$\text{Omega}(3, 4) = w(3) + c(3, 4) = [2, 4] + [3, 4] = [5, 8]$

$w(4) = \min \{ [10, 12], [2, 4], [5, 8] \}$

The evaluation of the minimum for this set of Omegas is rather straightforward, since all comparison sets are disjoint. If the Omega (1, 4) = [10, 12] and Omega(3, 4) = [5, 8]

were at their smallest values, {10 and 5}, they would not be less than Omega (2, 4). The true value of Omega(2, 4) will always be less than the Omega(1, 4) and Omega(3, 4). Therefore, the minimum is Omega(2, 4) = [2, 4]. Additionally, since Node 4 is the End Node in the network in Figure 11, w(4) = [2, 4] represents the shortest measure of performance from Node 1 to Node 4.

For the network shown in Figure 13, the values of w(1) = [0, 0], w(2) = [1, 2], and w(3) = [2, 4] are obvious. However, the value of w(4) is not obvious. To evaluate a minimum, the comparison of the Omega(j, 4) requires some elementary logic.

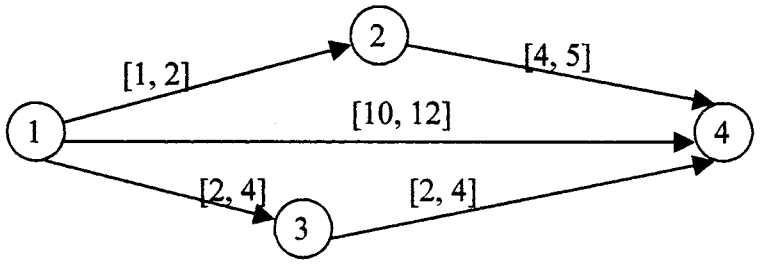


Figure 13: Transportation Network III: Nodes 1-4

Omega(1, 4) = [10, 12]

Omega(2, 4) = [1, 2] + [4, 5] = [5, 7]

Omega(3, 4) = [2, 4] + [2, 4] = [4, 8]

w(4) = min{ [10, 12], [5, 7], [4, 8] }

Figure 14 shows the values of Omega(1, 4), Omega(2, 4) and Omega(3, 4) relative to the real number line. In finding the minimum among these sets, the true minimum will be contained in some subset of the Omega values. It is important to differentiate among all possible cases of minimum of the Omega values. Specific cases occur at values between 4 and 5, between 5 and 7, between 7 and 8, between 8 and 10, and between 10 and 12.

- [4, 5]: Omega(3, 4) contains values in this interval
- [5, 7]: Omega(3, 4) and Omega(1, 4) contain values in this interval
- [7, 8]: Omega(1, 4) contains values in this interval
- [8, 10]: no Omega values are contained in this interval
- [10, 12]: Omega(2, 4) contains values in this interval

The real number line has been "cut" into a series of significant points; i.e. "cut points." The cut points of Node 4 are {4, 5, 7, 8, 10, 12}. For a non-trivial network, the case by case enumeration of the set of intervals is rather tedious. However, there are two significant cases of concern: the smallest possible minimum and the largest possible minimum. The smallest possible minimum would be the minimum of the lower bounds of each interval, which equals min{4, 5, 10} = 4. The largest possible minimum would be the minimum of upper bounds of each interval, which equals min{7, 8, 12} = 7. To appropriately described all possible events, the crucial cut points are the cut points contained in the interval between the smallest possible minimum and the largest possible minimum. Additionally, 5 is a cut point among the set of Omega's. The point, 5, is the

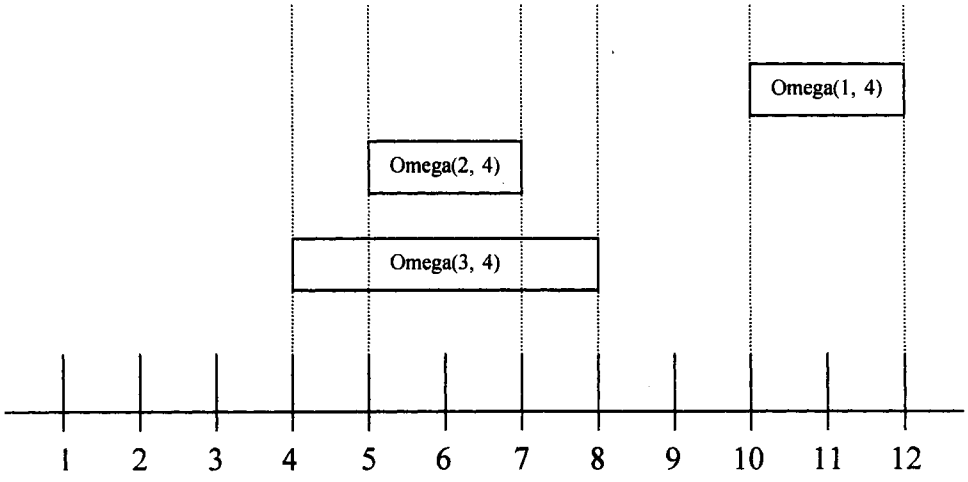


Figure 14: Omega Values

lower bound of Omega(2, 4). The point, 5, provides a point of differentiation between Omega(3, 4) and Omega(2, 4). Therefore, the crucial cut points are {4, 5, 7}, since no other minimum is contained inside any other cut-point. If the true minimum were between 4 and 5, this minimum value would lie in the interval of Omega(3, 4). If the true minimum were between 5 and 7 either Omega (3, 4) or Omega(2, 4) could contain the true minimum. If the true value of Omega(3, 4) is greater than 7, Omega(2, 4) will be the minimum. The true value of Omega(1, 4) is always greater than 7 and cannot be a minimum. Therefore, w(4) = min{ [10, 12], [5, 7], [4, 8] } = { $w_1(4)$, $w_2(4)$ } = { [4, 5], [5, 7] } and w(4) consists of 2 cases/threads.

The process of determining node thread values can be generalized as follows:

For any interval-network, the cut-points of a node are an ordered set of all upper and lower bounds of the Omega values. The cut-point set of Node k defines w(k). The two most significant cut-points are the minimum of the lower bounds of the Omega values and the minimum of the upper bounds of the Omega values. In the trivial example shown above, the only node containing multiple threads is node 4. The Omega values for node k are generated from the node*threads that precede node k. The minimum of the upper bounds of the Omega values (the stopping cut point) is the minimum from among the "last" node*thread of all preceding nodes. This distinction of choosing the minimum from among the "last" threads in necessary to ensure that the worst-case minimum is evaluated appropriately. Again, the stopping cut point signifies the largest possible minimum. This minimum must be among the largest from all predecessor nodes to appropriately analyze all situations. The minimum of the lower bounds of the Omega values is the starting cut point and the minimum of the upper bounds of the "last" Omega

values is the stopping cut point. These cut-points bound the intervals that the true minimum can be contained in and they are both significant in the algorithm.. No value greater than the stopping cut point can be the minimum value. That is, the stopping cut-point is the largest possible w(k) value. If there exist j and k such that Omega(j, k) $\subseteq$ $\{w_1(k) \cup w_2(k) \cup \cdots \cup w_p(k)\}$, this implies that for all i, $w_i(k) \leq w_p(k)$. Since each $w_i(k) = [CP_i, CP_{i+1}]$, if the stopping cut point is in the p^{th} position of the cut-point set, p-1 node threads exist.

Given Omega(j*a, k) = $[L_{jak}, U_{jak}]$, for all node*threads j*a that precede node k, the cardinality of node j is d. Let:

- CutPoints (k) = $\{ x \mid x = L_{jak} \} \cup \{ x \mid x = U_{jak} \} = \{ CP_1, CP_2, ..., CP_m \}$ such that $CP_{(i)} < CP_{(i+1)}$ for every i.
- StoppingCutPoint(k) = min$\{ x \mid x = U_{jdk} \}$, where d is the last position of each predecessor node.
- StoppingCutPointPosition = p. CutPoints (k) is an ordered set where CutPoints(k) = $\{ CP_1, CP_2, ..., CP_p$ = StoppingCutPoint(k) $\}$ and this is the ordered position of the stopping cut point. $w(n) = \{ w_1(n), w_2(n), ..., w_{p-1}(n) \}$ such that $w_1(n)$, $w_2(n), ..., w_{p-1}(n)$ are "nearly" disjoint sets. For all i, $w_i(n) \cap w_{i+1}(n) = \{b\}$ such that $w_i(n)=[a, b]$ and $w_{i+1}(n) = [b, c]$ where $a < c$. $w_1(n) = [CP_1, CP_2]$, $w_2(n) = [CP_2, CP_3], ..., w_i(n) = [CP_i, CP_{i+1}], ..., w_k(n) = [CP_{p-1}, CP_p]$. The cardinality of w(n) is $p - 1$.

Several Omega values and corresponding cut points are shown in Figure 15. CP1 is the smallest of the lower bounds of the Omega values and CP5 is the smallest of the

upper bounds of the Omega values generated by the "last" predecessor node*threads. CP5 is the stopping point, i.e., no values greater than CP5 can be the minimum.

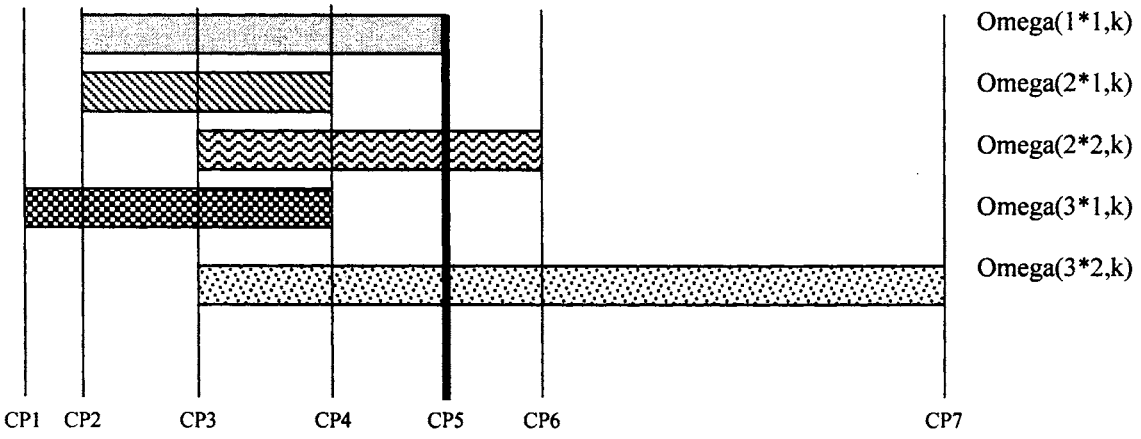


Figure 15: Cut Points and Omega Values

Appendix B shows the complete algorithm for finding the cut points and the stopping cut point. Appendix C shows the complete algorithm for ordering the cut points and finding the cut point position. Figure 16 shows the flow chart for Generate_w(k). This procedure occurs after the cut points are found and ordered.

In general, we are concerned with the number of node threads created by the algorithm. Due to the interval nature of the network, the number of node threads for the End Node can be very large. However, it is possible to find the maximum number of End Threads before implementing the algorithm. For any node k with τ_k Omega values being compared to find the node threads, the maximum number of node threads for node k is τ_k. The maximum number of Node k threads, $\mu_k = \sum_{(j,k)\in A} \mu_j$. For any network G satisfying the numbering node condition, $A = \{(a, b) \mid (a, b) \in A \rightarrow a < b\}$, the maximum number of nodes that precede node k is $k - 1$.

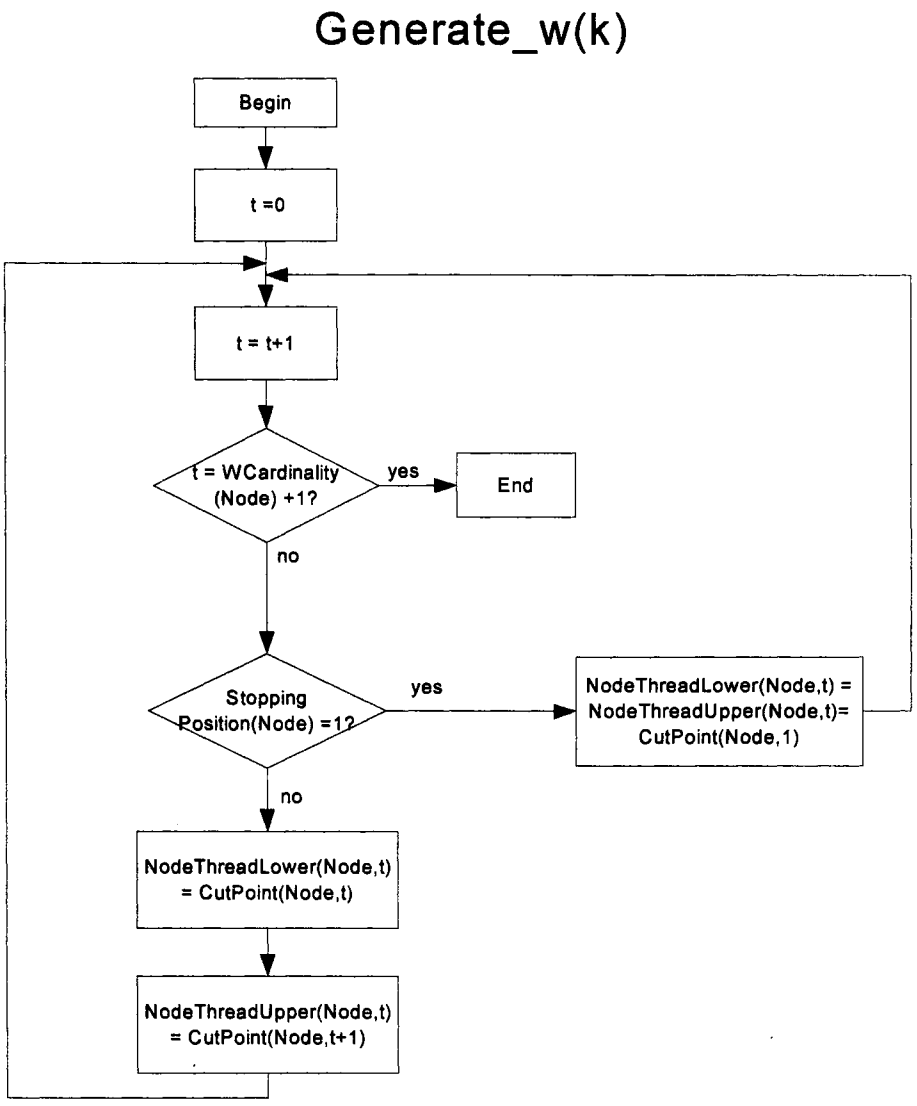


Figure 16: Flow Chart Generate_w(k)

Recall, the maximum number of threads for node k = μ_k. Thus, $\mu_1 = \mu_2 = 1$, $\mu_3 = \mu_1 + \mu_2 = 1 + 1 = 2$ and $\mu_4 = \mu_1 + \mu_2 + \mu_3 = 1 + 1 + 2 = 4$. The maximum number of threads for node k is $\mu_k = 2^{k-2}$.

Proof by induction:

$k = 3 = \mu_3 = 2^{3-2} = 2$

Let $\mu_k = \mu_1 + \mu_2 + \mu_3 + \ldots + \mu_{k-1} = 2^{k-2}$

$$\mu_{k+1} = \mu_1 + \mu_2 + \mu_3 + \ldots + \mu_{k-1} + \mu_k$$

$$\mu_{k+1} = (\mu_1 + \mu_2 + \mu_3 + \ldots + \mu_{k-1}) + \mu_k$$

$$\mu_{k+1} = \mu_k + \mu_k = 2\mu_k = 2*2^{k-2} = 2^{k-1}$$

Check Arcs

In traditional shortest path analysis, if w(b) – w(a) = c(a, b) then a shortest path for the network contains the arc (a, b). Due to the complexity of the interval-network, this condition may not be true. Furthermore, due to the nature of the interval-network, some nodes have more than one interval contained in w(n), i.e. there are multiple node threads. The condition for arc membership of a shortest path must be specific for each thread to yield a complete solution set.

For the network shown in Figure 17, Table III shows all combinations of differences in thread values, and the corresponding measure of performance along that arc. Given, $w_k(b) = [\hat{L}_{bk}, \hat{U}_{bk}]$, $w_j(a) = [\hat{L}_{aj}, \hat{U}_{aj}]$, and $c(a, b) = [L_{ab}, U_{ab}]$. An arc may be a member of a shortest path, if for some thread k of w(b) and some thread j of w(a) :

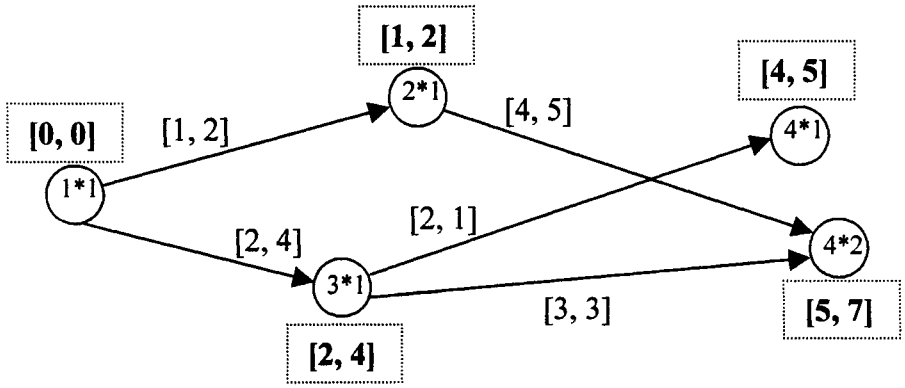


Figure 17: Node*Thread Network

$\hat{L}_{bk} - \hat{L}_{aj} \geq \hat{L}_{ab}$ and $\hat{U}_{bk} - \hat{U}_{aj} \leq \hat{U}_{ab}$ (Condition 1).

Note, this is a looser restriction than $w_k(b) - w_j(a) \subseteq c(a, b)$. For example, the arc from Node 3*Thread 1 to Node 4*Thread 1 satisfies condition 1, but not the proper subset condition. The need for this specific condition will be discussed later.

Table III
Arc Values

Arc	$w_k(b)-w_j(a)$ Lower	$w_k(b)-w_j(a)$ Upper	c(a, b) Lower	c(a, b) Upper	Condition 1 Satisfied?
$w_1(2)-w_1(1)$	1	2	1	2	Y
$w_1(3)-w_1(1)$	2	4	2	4	Y
$w_1(4)-w_1(1)$	4	5	10	12	N
$w_2(4)-w_1(1)$	5	7	10	12	N
$w_1(4)-w_1(2)$	3	3	4	5	N
$w_2(4)-w_1(2)$	4	5	4	5	Y
$w_1(4)-w_1(3)$	2	1	2	4	Y
$w_2(4)-w_1(3)$	3	3	2	4	Y

Each node*thread combination needs to be analyzed to fully describe all possible shortest paths. An amended network is constructed such that every Node a is replaced with Node a*l for all node threads l of Node a. A single arc from Node a to Node b is replaced by multiple arcs from Node a*l for all threads l of Node a and Node b*k for all threads k of Node b. Arcs that do not satisfy condition 1 are eliminated from the network.

For all arcs that satisfy condition 1, the measure of performance along the arc is given as c(Node a*Thread l, Node b*Thread k) = $[\hat{L}_{bk} - \hat{L}_{al}, \hat{U}_{bk} - \hat{U}_{al}]$. The amended network of Figure 12 is shown in Figure 17. The flow chart for Check_Possible_Arcs is shown in Figure 18.

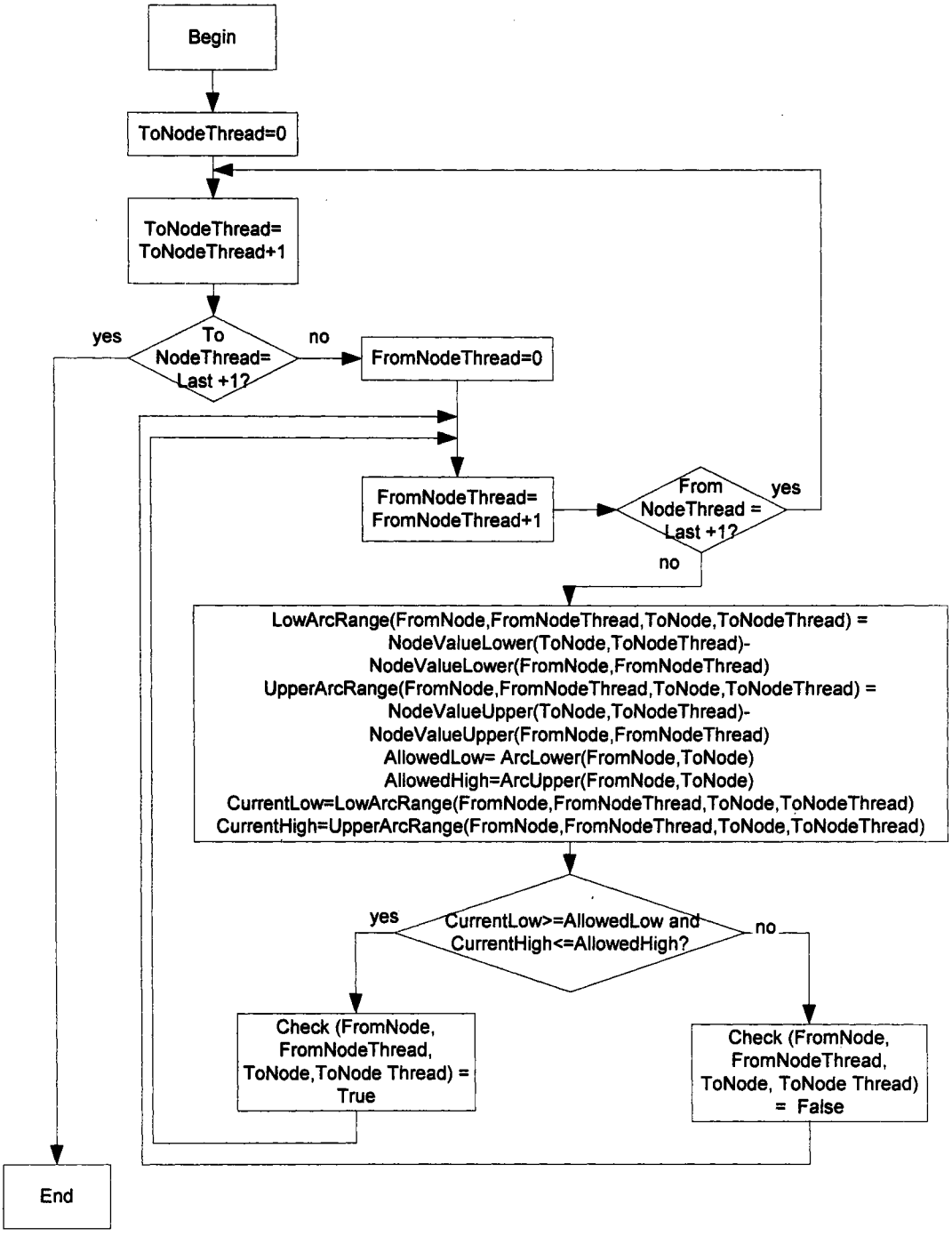


Figure 18: Flow Chart Check_Possible_Arcs

Find Paths

The algorithm performs all the operations necessary to build an amended network to find all the shortest paths of the network. The sub-procedure Find_Paths generates all possible shortest paths of the network given the node*thread and arc information. In traditional shortest path algorithms, the measure of performance along an arc for a specific shortest path is known before the algorithm is implemented and is therefore redundant. The value of measure of performance along the arcs in an interval-network is only bounded. Therefore, the c*(a, b) corresponding to all arcs (a, b) on the shortest paths are also described in the Find_Paths sub-procedure.

Given:

- $G = (N, A, c)$ and $N = \{1, 2, ..., n\}$
- $c(a, b) = [L_{ab}, U_{ab}]$
- $w_k(b) = [\hat{L}_{bk}, \hat{U}_{bk}]$ is the kth thread of Node b
- $w_j(a) = [\hat{L}_{aj}, \hat{U}_{aj}]$ is the jth thread of Node a

Let $G^* = (N^*, A^*, c^*)$ where

- $N^* = \{b^*k\}$; $b \in N = \{1, 2, ..., n\}$; $k = 1, 2, ...,$ cardinality of w(b). The cardinality of $N^* = \sum_{i=1}^{n} \text{cardinality of } w(i)$. K = cardinality of w(n)
- $A^* = \{(b^*k, a^*j) \mid \hat{L}_{bk} - \hat{L}_{aj} \geq \hat{L}_{ab} \text{ and } \hat{U}_{bk} - \hat{U}_{aj} \leq \hat{U}_{ab}.$ (Condition 1)$\}$
- $c^* = c^*(b^*k, a^*j) = [\hat{L}_{bk} - \hat{L}_{aj}, \hat{U}_{bk} - \hat{U}_{aj}] = [L_{bkaj}, U_{bkaj}]$.

An arc can only be traversed directly from Node i to Node j if the arc (i, j) exists, i.e., $c(i, j) < \infty$. A path moves from node to node by a specific arc connecting the two nodes. $\text{Node} \rightarrow \text{Arc} \rightarrow \text{Node} \rightarrow \cdots \rightarrow \text{Arc} \rightarrow \text{Node}$. As each arc (i, j), in the path is traversed, a

measure of performance, e.g., cost of c(i, j) is incurred. The objective is to find a path from N+ to N- with the shortest measure of performance, e.g. smallest cost.

Given a path P = (1*1, ..., b*k) and α = (b*k, a*j) such that $\alpha \in A^*$, path P* = (1*1, ..., b*k, a*j) is a shortest path with initial Node 1*1 and terminal Node a*j. The algorithm generates all shortest paths with initial Node 1*1 and terminal Node b*k for every b*k $\in$ N* using a iterative technique of adding nodes to existing shortest paths.

Initial paths are formed as paths with initial Node 1*1 and terminal Node b*k, for all b*k that are connected to Node 1*1 (the arc from 1*1 to b*k must exist). New paths are constructed as node*threads are added to previous shortest paths if the arc from the last position node*thread to b* k exists. The algorithm creates new paths by adding nodes to the end of existing paths and it is imperative to keep track of all nodes at the end of an existing path (Last Position (Path P)).

A complete set of shortest paths with initial Node 1*1, terminal Node n*j and cumulative length of the path $w_j(n) = [\hat{L}_{nj}, \hat{U}_{nj}]$ will be generated by the algorithm. Figure 19 shows the flow chart for Find_Paths. Below is each step of the algorithms procedure for finding the shortest path of the network shown in Figure 17.

Last Position of existing paths = {1*1}

To Node = 2 Is 2*1 connected to 1*1? Yes. Node 2, Path 1= 1*1 to 2*1

Last Position of existing paths = {1*1, 2*1}

To Node = 3 Is 3*1 connected to 1*1? Yes Node 3, Path 1 = 1*1 to 3*1

Is 3*1 connected to 2*1? No

Last Position of existing paths = {1*1, 2*1, 3*1}

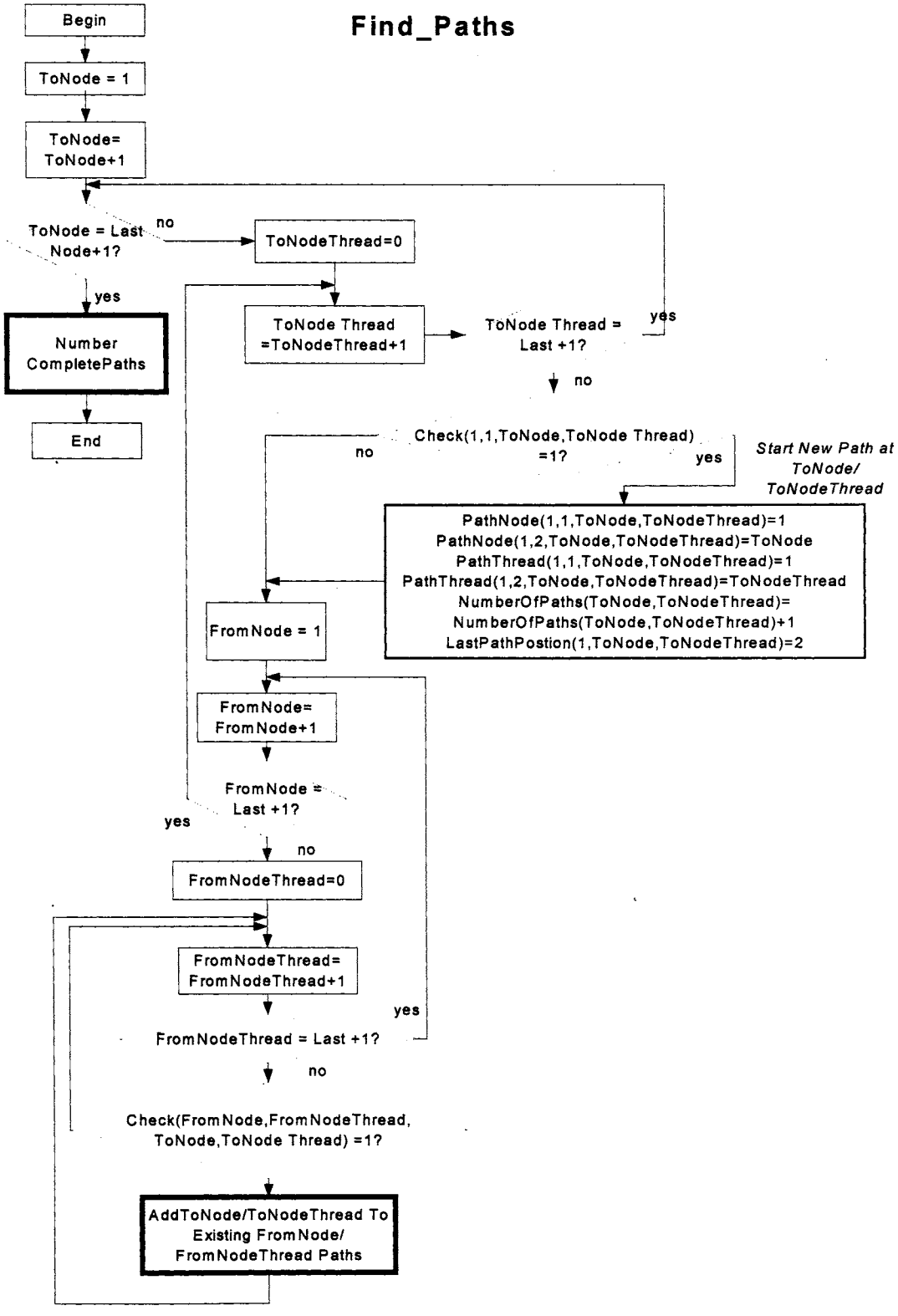


Figure 19: Flow Chart Find_Paths

To Node = 4 Is 4*1 connected to 1*1? No

Is 4*1 connected to 2*1? No

Is 4*1 connected to 3*1? Yes Node 4, Path 1 = 1*1 to 3*1 to 4*1

Is 4*2 connected to 1*1? No

Is 4*2 connected to 2*1? Yes Node 4, Path 2 = 1*1 to 2*1 to 4*2

Is 4*2 connected to 3*1? Yes Node 4, Path 3 = 1*1 to 3*1 to 4*2

Last Position of existing paths = {1*1, 2*1, 3*1, 4*1, 4*2}

To Node = 4 → End.

Change Arc On Path

Due to the structure of condition 1, it is possible that a corresponding measure of performance along an arc has a lower bound greater than the upper bound. The sub-procedure Change_Arc_On_Path, shown in Figure 20, corrects these possible inconsistencies in the output. After the Find_Paths sub-procedure is completed, the algorithm would have generated 3 shortest paths for the network in Figure 12.

Length Path 1 = 1*1 to 3*1 to 4*1 = [2, 4] + [2, 1] = [4, 5]

Length Path 2= 1*1 to 2*1 to 4*2 = [1, 2] + [4, 5] = [5, 7]

Length Path 3 = 1*1 to 3*1 to 4*2 = [2, 4] + [3, 3] = [5, 7]

Since Node 4 has two node threads, there are two possible shortest path lengths ([4, 5] and [5, 7]). Path 2 and Path 3 both have cumulative length of [5, 7] and Path 1 ([2, 4] + [2, 1]) has cumulative length of [4, 5]. Note that the arc from 3*1 to 4*1 satisfies Condition 1 but not the subset condition. If it had not been considered a "good"

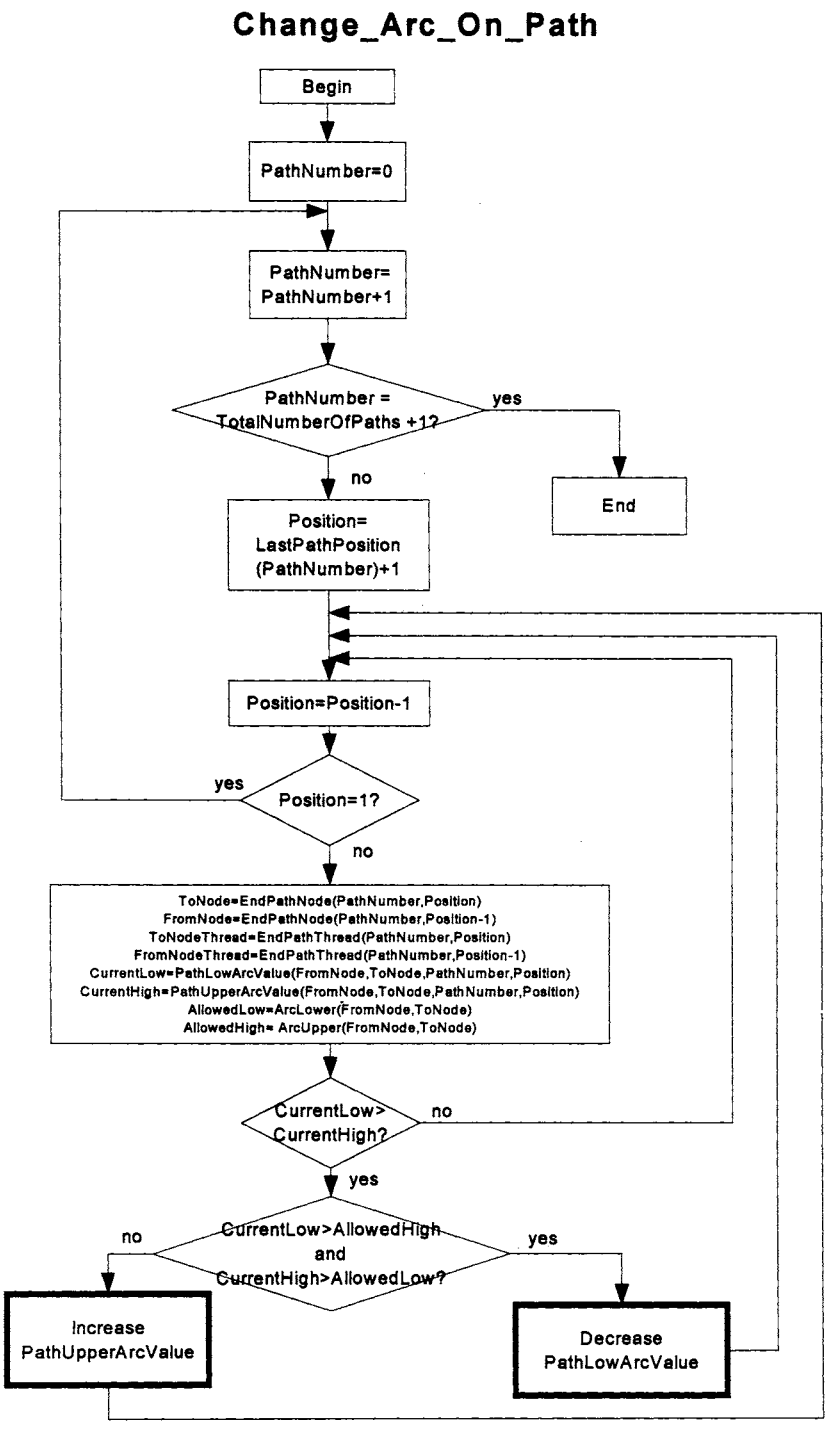


Figure 20: Flow Chart Change_Arc_On_Path

arc, this path would not have been enumerated as a shortest path. The primary objective is to list all possible paths whose cumulative lengths are a thread of the End Node. Path 1 satisfies this condition. The value of [2, 1] for the arc from 3*1 to 4*1 is not a properly defined interval-value. The corresponding measures of performance along the arcs are necessary to formulate a complete path description. There are an infinite number of combinations of 1*1 to 3*1 and 3*1 to 4*1, which would yield the [4, 5] distance. Recall that the arc range from 1*1 to 3*1 = [2, 4] and 3*1 to 4*1 = [2, 3]. Below are a small sample of combinations which yield the cumulative path distance of [4, 5]:

- 1*1 to 3*1 = [2, 3] 3*1 to 4*1 = [2, 2]
- 1*1 to 3*1 = [2, 2.5] 3*1 to 4*1 = [2, 2.5]
- 1*1 to 3*1 = [2, 2] 3*1 to 4*1 = [2, 3]

The algorithm finds only one alternative that will satisfy the cumulative path length condition, since the primary objective is the generation of the shortest paths. The exact values for the measure of performance along an arc is qualitative by nature and does not need to be known. However, a measure of performance such that the lower bound is greater than the upper bound is counter-intuitive and doesn't yield a meaningful path description.

The sub-procedure Change_Arc_On_Path changes the values along the arcs to yield all arc values such that:

Given

- Path P* = (1*1, ..., b*k, a*j, ..., n*m)
- $w_m(n) = [\hat{L}_{nm}, \hat{U}_{nm}]$ = [End Node thread lower bound, End Node thread upper bound]

- $c(b, a) = [L_{ba}, U_{ba}]$ = [arc lower bound, arc upper bound]
- $c^* = c^*(b^*k, a^*j) = [L_{bkaj}, U_{bkaj}]$ = [arc thread lower bound, arc thread upper bound] for every Path P*, $A^* = (b^*k, a^*j) \in P^*$, and $c^*(b^*k, a^*j)$
- $L_{bkaj} \leq U_{bkaj}$ (The arc thread lower bound is less than the arc thread upper bound).
- $L_{bkaj} \geq L_{ab}$ (The arc thread lower bound is greater than or equal to the arc lower bound).
- $U_{bkaj} \leq U_{ab}$ (The arc thread upper bound is less than or equal to the arc upper bound).
- $\sum_{(b/k,a/j)\in P^*} L_{bkaj} = \hat{L}_{nm}$ (The lower cumulative path length equals the path thread lower bound).
- $\sum_{(b/k,a/j)\in P^*} U_{bkaj} = \hat{U}_{nm}$ (The upper cumulative path length equals the path thread upper bound).

The arc thread values along each path are changed such that the lower arc thread bound is less than the upper arc thread bound without loosing the integrity of the cumulative path length. This adjustment is accomplished by decreasing and increasing arc thread values simultaneously. If it is necessary to increase an arc thread's lower value, a decrease of the same magnitude in another arc thread's lower value must accompany the increase. Similarly, if it is necessary to decrease an arc thread's upper value, an increase of the same magnitude in another arc thread's upper value must accompany the decrease. The algorithms for sub-procedures Decrease_Path_Low_Arc_ Value and Increase_Path_High_Arc_Value are in Appendix D and Appendix E, respectfully.

For example, the arc from 3*1 to 4*1 in Path 1 from above has a value [2, 1]. The arc from 3 to 4 has value of [2, 4]. For this example, it is necessary to increase the arc upper value by at least one unit.. However, the upper value of another arc thread along the path must be decreased by the same value. The arc adjacent to (3*1, 4*1) is (1*1, 3*1). Since the arc from 1 to 3 has a value of [2, 4] and the arc thread value is [2, 4], the upper arc thread value can be decreased as much as 2 units. Since the necessary change is one unit and 2 units are available to be changed, the upper bound of arc (3*1, 4*1) is increased one unit and the upper bound of (1*1, 3*1) is decreased one unit. After creating the new arc thread values, the length of Path 1 = 1*1 to 3*1 to 4*1 = [2, 3] + [2, 2] = [4, 5].

For longer paths, the amount available in the adjacent arc may not be as large as the needed change. In this instance, the largest change possible is made in each arc along the path until the required change is accomplished.

After the sub-procedure Change_Arc_On_Path is complete, the algorithm has generated all possible shortest paths for the interval-network and each arc value along the path is a subset of the interval defined in the problem statement. The development of the methodology for the analysis of the set of shortest paths will be discussed in Chapter V.

CHAPTER IV

ALGORITHM RESULTS

Introduction

This chapter will show several examples of output from the algorithm discussed in Chapter III. It is impossible to give an exhaustive list of the types of shortest path problems. However, the examples shown in this chapter should give the reader a basic overview of the situations that result when implementing this algorithm. The algorithm was coded in Visual Basic [11] with Microsoft Excel [10] as the interface. The Visual Basic code is in Appendix A. Node data and arc measure of performance bounds are the only required input for this algorithm. The lower and upper measure of performance along the arcs are written in matrix form in a Microsoft Excel [10] Spreadsheet. If the arc connecting two nodes does not exist, the lower and upper arc bounds are infinity (infinity has been defined as 9999 in this program). Additionally, all significant output data is written to the Microsoft Excel [10] Spreadsheet.

The first output of the algorithm is the Node Threads or $\{w_i(k)\}$ for all nodes k. The quantity of Node Threads is a major contributor to the complexity of the algorithm. The Node Threads are generated in the first stages of the algorithm. The number of iterations remaining in the algorithm is a direct result of the number of Node Threads. Additionally, the size of the shortest path solution set is directly related to the number of Node Threads. A large number of Node Threads generally occurs in a network with many nodes and/or large ranges between the bounds of the measure of performance. As the range between the bounds increases, the network resembles an unbounded network.

As the size of the network approaches an unbounded network, the set of shortest paths from Node 1 to Node n approaches the set of all paths from Node 1 to Node n.

The most significant output of the algorithm is the set of all paths in Node*Thread notation. The shortest paths are based on End Thread information. End Threads are a specific set of Node Threads for the destination node of the shortest path. The End Threads specify the possible shortest path distances. The path information consists of the nodes traversed in the path, the shortest path distance, and the measure of performance along the arcs in the path which correspond to that shortest distance. The generation of the shortest path is only possible after the algorithm has checked all arcs against specific conditions to find the set of possible arcs. Additionally, the measure of performance along the arcs is corrected if the lower bound of the measure of performance is greater than the upper bound of the measure of performance.

10-Node Traditional Networks

The algorithm is designed to yield all possible shortest paths of a network with interval-valued measures of performance . However, the algorithm will yield the shortest path of a network with single-valued measures of performance. If a measure of performance along an arc is a constant, it is described by an interval such that the upper and lower bounds are equal. Figure 21 shows a 10-Node network with constant-valued measures of performance. Table IV shows the Node Thread values $\{w_k(b) = [\hat{L}_{bk}, \hat{U}_{bk}]$ = [Lower Node Thread, Upper Node Thread]} for Node b, b = 2, 3,...,10. These are the exact w(k) values that would result by implementing Dijkstra's method. Table V shows all arcs of an amended network of the original network given in Figure 21. In the

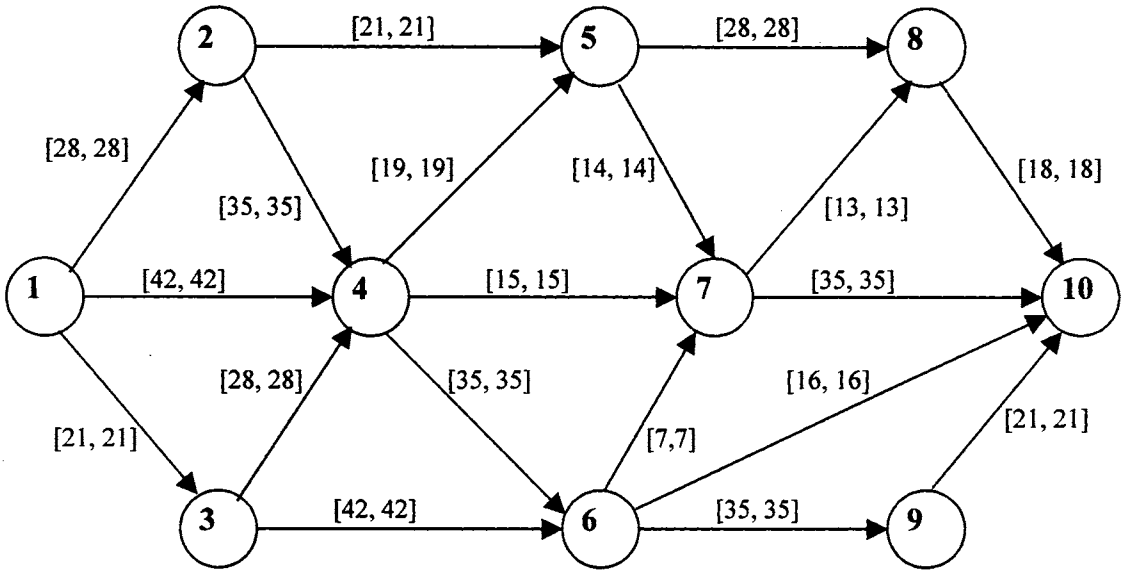


Figure 21: 10-Node Network I

amended network, each single node is replaced by all node*threads for that node. The arcs in Table V are denoted by the originating node*thread (TO NODE) and terminating node*thread (FROM NODE). Only the arcs that satisfy Condition 1 may be members of a shortest path. However, many of the arcs that could be members of the shortest path may not actually be members of the shortest path. Basically, an arc is a possible member of a shortest path if the arc "generated" a specific Node Thread value. The Node Thread value, w(k), is a minimum from among a set of w(a) + c(a, k) of all Node a's proceeding Node k. Table VI shows the one shortest path of the network in Figure 21 (Node 1 → Node 3 → Node 6 → Node 10). The existence of only one shortest path is not a surprise, considering the single-valued measures of performance along the arcs.

Table IV
Node Threads for 10-Node Network I

Node	Thread	2	3	4	5	6	7	8	9	10
Lower	1	28	21	42	49	63	57	70	98	79
Upper	1	28	21	42	49	63	57	70	98	79

Table V
Possible Arcs for 10-Node Network I

TO NODE	11	11	11	21	21	31	31	41	41	41	51	51	61	61	61	71	71	81	91
FROM NODE	21	31	41	41	51	41	61	51	61	71	71	81	71	91	101	81	101	101	101
CONDITION 1 SATISFIED?	Y	Y	Y	N	Y	N	Y	N	N	Y	N	N	N	Y	Y	Y	N	N	N

Table VI
Path Threads for 10-Node Network I

Arc	Lower Bound	Lower Value	Upper Value	Upper Bound		Node* Thread	Node* Thread		Lower End Thread	Upper End Thread
Arc1	21	21	21	21		11	31			
Arc2	42	42	42	42		31	61			
Arc3	16	16	16	16		61	101		79	79

The primary objective of this research project was the generation of shortest paths when the exact measures of performance along the arcs were unknown. The algorithm described in Chapter III finds all shortest paths of a network whose measures of performance along the arcs are bounded above and below. One is tempted to find the shortest path of a network with constant measures of performance at each of the bounds.

For instance, the network in Figure 21 may be the lower bounds of the measures of performance along the arcs. Figure 22 shows a constant network such that each measure of performance along an arc is four units greater than the network in Figure 21. This would describe the upper bounds of the measures of performance along the arcs if the measures of performance of the network in Figure 21 could deviate as much as four units higher than the values seen in Figure 21. Table VII shows the Node Thread values for Node 2 through Node 10 of the network in Figure 22. Table VIII shows that the identical shortest path to the network for Figure 22 that was obtained for the network in Figure 21. The only difference is the measure of performance along the arcs.

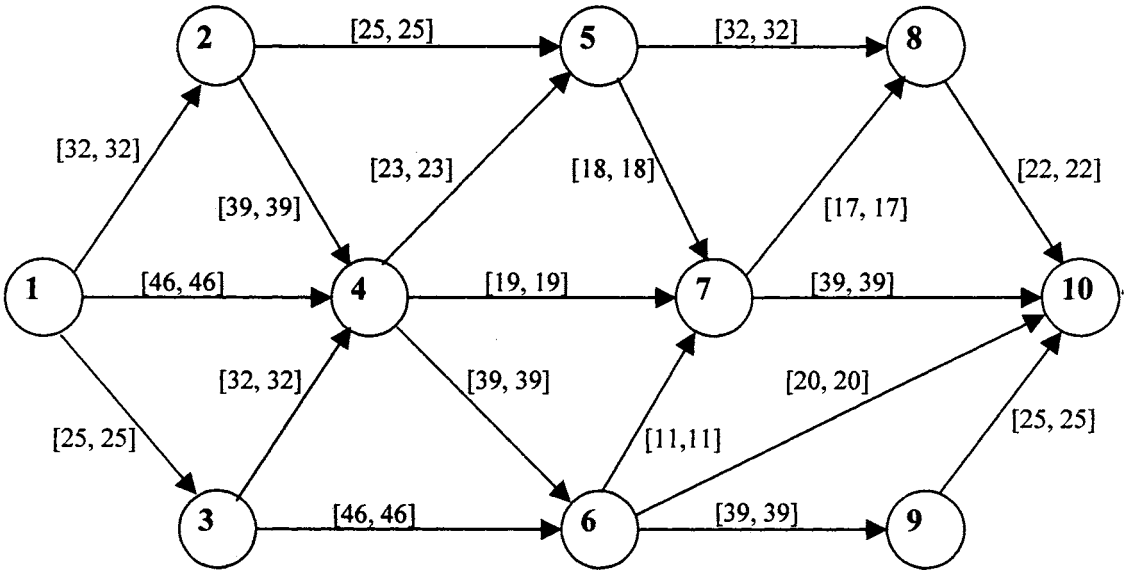


Figure 22: 10-Node Network II

Table VII
Node Threads for 10-Node Network II

Node	Thread	2	3	4	5	6	7	8	9	10
Lower	1	40	30	60	70	90	80	100	140	110
Upper	1	40	30	60	70	90	80	100	140	110

Table VIII
Path Threads for 10-Node Network II

Arc	Lower Bound	Lower Value	Upper Value	Upper Bound		Node* Thread	Node* Thread		Lower End Thread	Upper End Thread
Arc1	30	30	30	30		11	31			
Arc2	60	60	60	60		31	61			
Arc3	20	20	20	20		61	101		110	110

10-Node Network III

Figure 23 shows a 10-Node network with interval-valued arc lengths which contains the constant measures of performance of the network in Figure 21 as the lower bounds for the measure of performance and the constant measures of performance in Figure 22 as the upper bounds for the measures of performance. The interval-values complicate the

problem as all points contained between the bounds are a possible measure of performance along an arc.

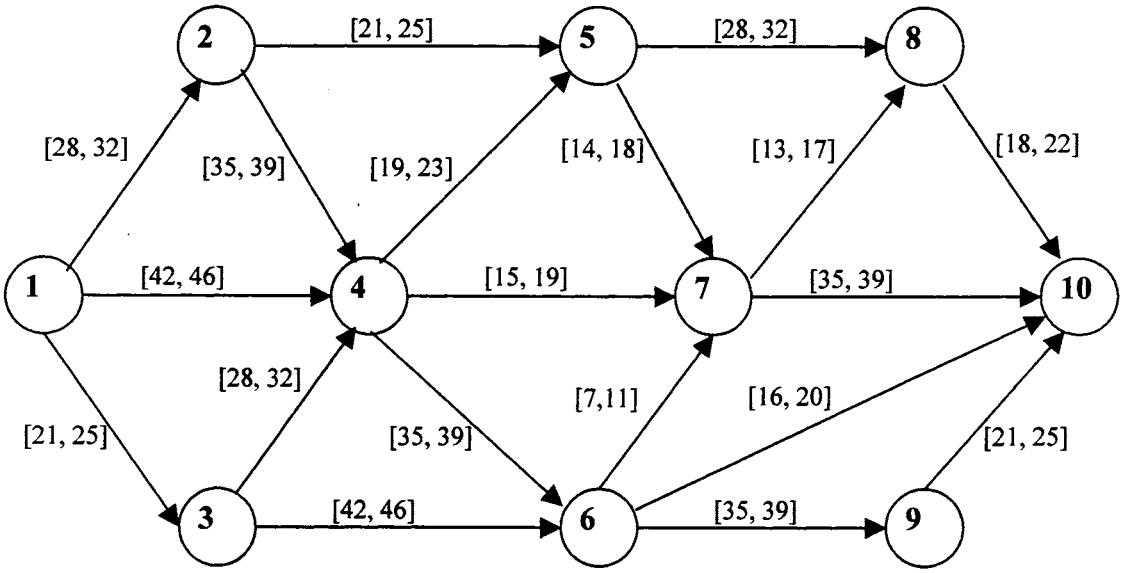


Figure 23: 10-Node Network III

Table IX shows the Node Thread values $\{w_k(b) = [\hat{L}_{bk}, \hat{U}_{bk}] = [\text{Lower Node Thread, Upper Node Thread}]\}$ for Node b, b = 2, 3,...,10 for the network in Figure 23. For any 10-Node network, the number of End Threads, $w(10) = \{w_i(10)\}$ can be extremely large. Specifically for this network, the number of End Threads could approach $2^{10-2} = 2^8 = 256$. However, there are only two End Threads for the network in Figure 23. The four unit interval-range, i.e., the upper bound minus the lower bound, along each arc contributes to the small number of End Threads.

Table IX
Node Threads for 10-Node Network III

Node	Thread	2	3	4	5	6	7	8	9	10
Lower	1	28	21	42	49	63	57	70	98	79
Upper	1	32	25	46	57	71	63	76	110	88
Lower	2						63	76		88
Upper	2						65	77		91
Lower	3							77		
Upper	3							80		
Lower	4							80		
Upper	4							82		

However in general, the basic structure of the network and the proximity among the measures of performance along the arcs contributes substantially to the number of End Threads and subsequently the number of shortest paths. Measure of performance along the arcs that are near one another creates Omega values with a greater number of intersections and therefore a greater number of Node Threads.

These two End Threads represent the two possible sets of cumulative shortest path lengths, [79, 88] and [88, 91]. Basically, the shortest path will be between 79 and 91 units. However, the interval from 79 to 91 is partitioned into two sets, to give us specific information on which shortest path will yield the specific cumulative path lengths. For the network in Figure 23, the shortest paths and corresponding measures of performance along the arcs which yield that shortest path length are shown in Table X. In Table X the measure of performance along any arc(a, b) is calculated as w(b) – w(a).

Table X
Uncorrected Path Threads 10 Node Network III

Path	Arc	Lower Bound	Lower Value	Upper Value	Upper Bound		Node* Thread	Node* Thread		End Thread	
										Lower	Upper
1	1	21	21	25	25		11	31			
	2	42	42	46	46		31	61			
	3	16	16	17	20		61	101		79	88
2	1	21	21	25	25		11	31			
	2	42	42	46	46		31	61			
	3	16	25	20	20		61	102		88	91
3	1	42	42	46	46		11	41			
	2	15	15	17	19		41	71			
	3	13	13	13	17		71	81			
	4	18	18	15	22		81	102		88	91

The arc measures of performance listed in Table X contain arc values such that the lower bound is greater than the upper bound. The sub-procedure Change_Arc_On_Path corrects the measure of performance along arcs as shown in Table XI. The output shown

in Table XI will be further analyzed according to the output methodology described in Chapter V.

Table XI
Corrected Path Threads 10 Node Network III

Path	Arc	Lower Bound	Lower Value	Upper Value	Upper Bound		Node* Thread	Node* Thread		End Thread	
										Lower	Upper
1	1	21	21	25	25		11	31			
	2	42	42	46	46		31	61			
	3	16	16	17	20		61	101		79	88
2	1	21	22	25	25		11	31			
	2	42	46	46	46		31	61			
	3	16	20	20	20		61	102		88	91
3	1	42	42	45	46		11	41			
	2	15	15	15	19		41	71			
	3	13	13	13	17		71	81			
	4	18	18	18	22		81	102		88	91

10-Node Network IV

Figure 24 shows another 10-Node network identical to the network in Figure 21. The lower bounds for the measures of performance are identical to the network in Figure 21, but the interval-ranges are varying quantities. The smallest range is three units, [7, 10],

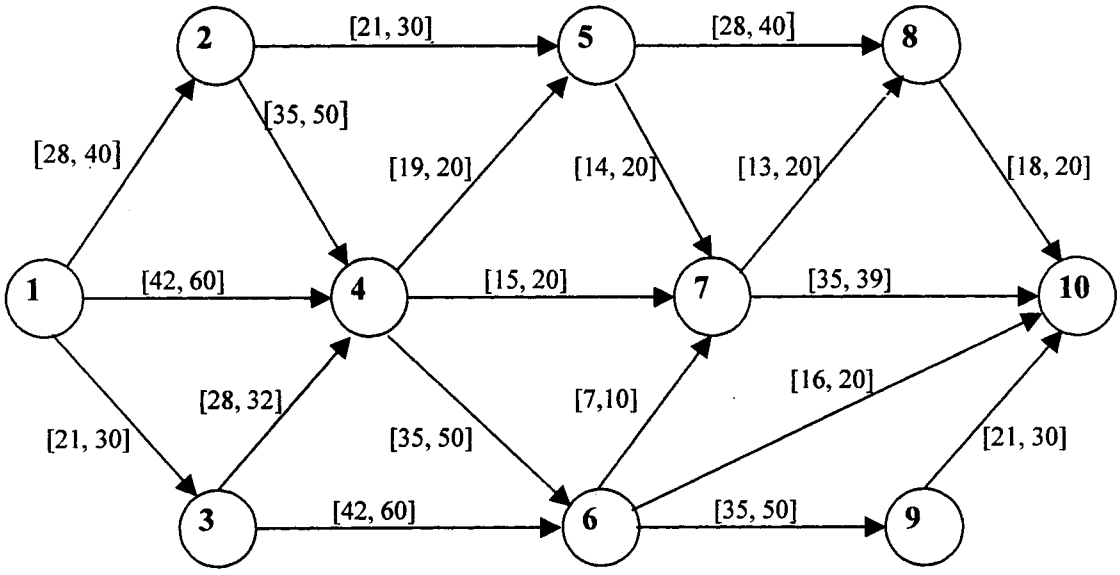


Figure 24: 10-Node Network IV

and the largest range is 18 units, [42, 60]. Table XII shows the $\{w_k(b) = [\hat{L}_{bk}, \hat{U}_{bk}] =$ [Lower Node Thread, Upper Node Thread]$\}$ for Node b, b = 2, 3,..., 10 for the network in Figure 24.

Table XII
Node Threads for 10 Node Network IV

Node	Thread	2	3	4	5	6	7	8	9	10
Lower	1	28	21	42	49	63	57	70	98	79
Upper	1	40	30	49	61	77	63	76	112	88
Lower	2			49	61	77	63	76	112	88
Upper	2			60	68	84	64	77	119	92
Lower	3				68	84	64	77	119	92
Upper	3				69	90	69	82	127	93
Lower	4				69		69	82	127	93
Upper	4				70		70	83	134	94
Lower	5						70	83	134	94
Upper	5						75	84	140	95
Lower	6						75	84		95
Upper	6						80	88		96
Lower	7							88		96
Upper	7							89		97
Lower	8							89		97
Upper	8							90		98
Lower	9							90		98
Upper	9							95		99
Lower	10							95		99
Upper	10							96		100
Lower	11							96		100
Upper	11							97		101
Lower	12							97		101
Upper	12							100		102
Lower	13									102
Upper	13									103
Lower	14									103
Upper	14									104
Lower	15									104
Upper	15									105
Lower	16									105
Upper	16									106
Lower	17									106
Upper	17									107
Lower	18									107
Upper	18									108
Lower	19									108
Upper	19									109
Lower	20									109
Upper	20									110

As expected, there are more End Threads for the network in Figure 24 than the network in Figure 23. Table XII displays the 20 End Threads for the network shown in Figure 24. This is not near the maximum of 256. The algorithm generated 538 shortest paths for the network in Figure 24. The 538 shortest paths for this network are shown in Appendix F. These shortest paths are described by the node*threads that the paths traverse and the measures of performance along the arcs that are necessary to yield that shortest path distance. The shortest path output for the network in Figure 24 will be further analyzed according to the output methodology described in Chapter V.

7-Node Network

A 7-Node network is shown in Figure 25. The structure of this network is relatively simple. Each node has less than four entering arcs. Additionally the interval-range along each arc is less than five units. Table XIII shows the $\{w_k(b) = [\hat{L}_{bk}, \hat{U}_{bk}] = [\text{Lower Node Thread, Upper Node Thread}]\}$ for Node b, b = 2, 3,...,7 for the network in Figure

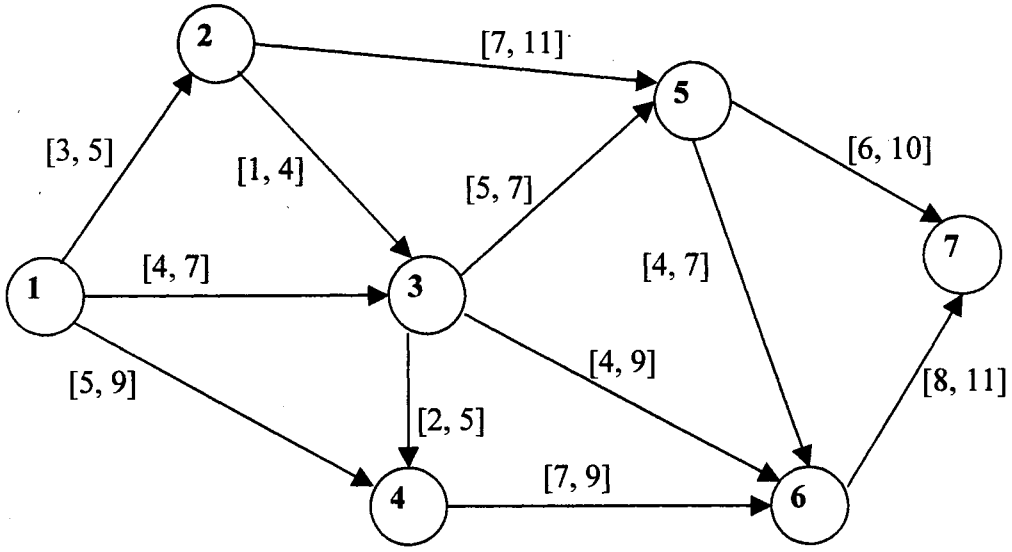


Figure 25: 7-Node Network

Table XIII
Node Threads for 7-Node Network

Node	Thread	2	3	4	5	6	7
Lower	1	3	4	5	9	8	15
Upper	1	5	7	6	10	12	16
Lower	2			6	10		16
Upper	2			9	14		20
Lower	3						20
Upper	3						23

25. There are only three End Threads for the network in Figure 25. However, there are 20 shortest paths for the network in Figure 25. This is a relatively large number of shortest paths even though there are only three End Threads. However, the values of the measures of performance along the arcs are relatively close to one another. This situation yields a network such that a large portion of the arcs satisfy Condition 1 and therefore can be members of the shortest path. Specifically, as seen in Table XIV, 16 of the 25 arcs connecting node*threads can be members of the shortest path. Table XV shows the 20 shortest paths generated by the algorithm The shortest path output for the network in Figure 25 will be further analyzed according to the output methodology described in Chapter V.

Table XIV
Possible Arcs for 7-Node Network

TO NODE	11	11	11	11	21	21	21	31	31	31	31	31	41
FROM NODE	21	31	41	42	31	51	52	41	42	51	52	61	61
CONDITION 1 SATISFIED?	Y	Y	Y	Y	Y	N	Y	N	Y	Y	Y	Y	N
TO NODE	42	51	52	51	52	51	52	51	52	61	61	61	
FROM NODE	61	61	61	71	71	72	72	73	73	71	72	73	
CONDITION 1 SATISFIED?	N	N	N	Y	N	Y	Y	N	Y	N	Y	Y	

Table XV
Path Threads for 7 Node Network

Path	Arc	Lower Bound	Lower Value	Upper Value	Upper Bound		Node* Thread	Node* Thread		End Thread	
										Lower	Upper
1	1	4	4	5	7		11	31			
	2	5	5	5	7		31	51			
	3	6	6	6	10		51	71		15	16
2	1	3	3	4	5		11	21			
	2	1	1	1	4		21	31			
	3	5	5	5	7		31	51			
	4	6	6	6	10		51	71		15	16
3	1	4	4	5	7		11	31			
	2	5	5	5	7		31	51			
	3	6	7	10	10		51	72		16	20
4	1	3	3	4	5		11	21			
	2	1	1	1	4		21	31			
	3	5	5	5	7		31	51			
	4	6	7	10	10		51	72		16	20
5	1	3	3	5	5		11	21			
	2	7	7	9	11		21	52			
	3	6	6	6	10		52	72		16	20
6	1	4	4	7	7		11	31			
	2	5	6	7	7		31	52			
	3	6	6	6	10		52	72		16	20
7	1	3	3	5	5		11	21			
	2	1	1	2	4		21	31			
	3	5	6	7	7		31	52			
	4	6	6	6	10		52	72		16	20
8	1	4	4	7	7		11	31			
	2	4	4	5	5		31	61			
	3	8	8	8	11		61	72		16	20
9	1	3	3	5	5		11	21			
	2	1	1	2	4		21	31			
	3	4	4	5	5		31	61			
	4	8	8	8	11		61	72		16	20
10	1	3	3	5	5		11	21			
	2	7	7	8	11		21	52			
	3	6	10	10	10		52	73		20	23
11	1	4	4	7	7		11	31			
	2	5	6	6	7		31	52			
	3	6	10	10	10		52	73		20	23
12	1	3	3	5	5		11	21			
	2	1	1	2	4		21	31			
	3	5	6	6	7		31	52			
	4	6	10	10	10		52	73		20	23
13	1	4	4	7	7		11	31			
	2	4	5	5	5		31	61			
	3	8	11	11	11		61	73		20	23
14	1	3	3	5	5		11	21			
	2	1	1	2	4		21	31			
	3	4	5	5	5		31	61			
	4	8	11	11	11		61	73		20	23

6-Node Network

As mentioned in Chapter II, Okada and Soper [11] have modeled a network using trapezoidal fuzzy arc lengths. These arc values are more limiting than interval-valued arcs. The network in Figure 26 contains interval-values similar to the trapezoid numbers introduced by Okada and Soper.

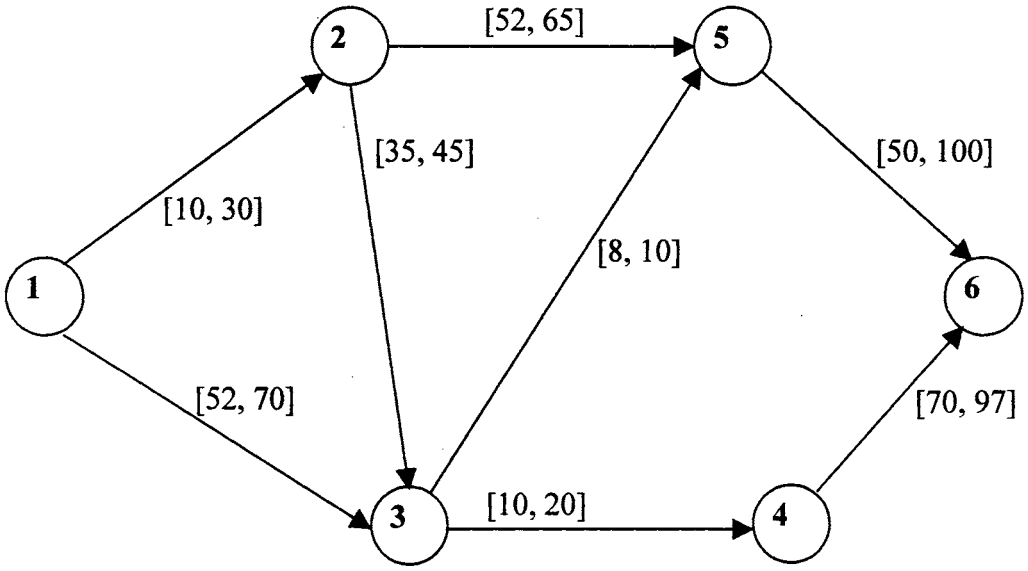


Figure 26: 6 Node Network

For the measure of performance along the arcs, each trapezoidal number [a, b, c, d] is translated into the interval-value [a − c, b + d]. This network is NOT identical to the network introduced by Okada and Soper. However, this modification to the Okada and Soper network is the best opportunity to compare the results of the algorithm discussed in Chapter III with results from the literature. Table XVI and XVII show the Node Threads and Path Threads, respectively, for the network shown in Figure 26. The shortest path output for the network in Figure 26 will be further analyzed according to the output methodology described in Chapter V.

Table XVI
Node Threads for 6-Node Network

Node	Thread	2	3	4	5	6
Lower	1	10	45	55	53	103
Upper	1	30	52	62	60	110
Lower	2		52	62	60	110
Upper	2		70	72	62	112
Lower	3			72	62	112
Upper	3			90	80	125
Lower	4					125
Upper	4					132
Lower	5					132
Upper	5					142
Lower	6					142
Upper	6					159
Lower	7					159
Upper	7					160
Lower	8					160
Upper	8					162
Lower	9					162
Upper	8					169
Lower	10					169
Upper	10					180

The six networks shown as examples in this chapter were developed with different characteristics in order to give a brief overview of the results of the Avery Shortest Path Algorithm. The first output of the algorithm was the Node Threads or $\{w_i(k)\}$ for all nodes k. The number of Node Threads was a major contributor to the complexity of the algorithm and the set of 6 shortest path problems yielded 1, 1, 2, 3, 10, and 20 End Threads, respectively.

The most significant output of the algorithm was the set of all shortest paths, Path Threads. Each shortest path was described by the Node*Threads traversed along the path, the corresponding End Thread (the cumulative path length), and the measure of performance along each arc, which yielded the specific End Thread. The set of 6 shortest path problems yielded 1, 1, 3, 14, 80, and 538 shortest paths, respectively. The different number of End Threads and Path Threads was a result of the span of the intervals, the

Table XVII:
Path Threads for 6-Node Network

Path	Node Threads					Path	Node Threads				
1	101	201	301	501	601	41	101	201	302	502	606
2	101	201	301	501	602	42	101	201	503	606	
3	101	201	301	502	602	43	101	302	503	606	
4	101	302	502	602		44	101	201	302	503	606
5	101	201	302	502	602	45	101	201	301	402	607
6	101	201	301	501	603	46	101	302	402	607	
7	101	201	301	502	603	47	101	201	302	402	607
8	101	302	502	603		48	101	302	403	607	
9	101	201	302	502	603	49	101	201	302	403	607
10	101	201	503	603		50	101	201	301	501	607
11	101	302	503	603		51	101	201	301	502	607
12	101	201	302	503	603	52	101	302	502	607	
13	101	201	301	401	604	53	101	201	302	502	607
14	101	201	301	501	604	54	101	201	503	607	
15	101	201	301	502	604	55	101	302	503	607	
16	101	302	502	604		56	101	201	302	503	607
17	101	201	302	502	604	57	101	201	301	402	608
18	101	201	503	604		58	101	302	402	608	
19	101	302	503	604		59	101	201	302	402	608
20	101	201	302	503	604	60	101	302	403	608	
21	101	201	301	401	605	61	101	201	302	403	608
22	101	201	301	402	605	62	101	201	301	502	608
23	101	302	402	605		63	101	302	502	608	
24	101	201	302	402	605	64	101	201	302	502	608
25	101	201	301	501	605	65	101	201	503	608	
26	101	201	301	502	605	66	101	302	503	608	
27	101	302	502	605		67	101	201	302	503	608
28	101	201	302	502	605	68	101	201	301	402	609
29	101	201	503	605		69	101	302	402	609	
30	101	302	503	605		70	101	201	302	402	609
31	101	201	302	503	605	71	101	302	403	609	
32	101	201	301	401	606	72	101	201	302	403	609
33	101	201	301	402	606	73	101	201	503	609	
34	101	302	402	606		74	101	302	503	609	
35	101	201	302	402	606	75	101	201	302	503	609
36	101	302	403	606		76	101	302	403	610	
37	101	201	302	403	606	77	101	201	302	403	610
38	101	201	301	501	606	78	101	201	503	610	
39	101	201	301	502	606	79	101	302	503	610	
40	101	302	502	606		80	101	201	302	503	610

relationship among the measures of performance along the arcs, and the basic structure of the network. The information describing the Path Threads was necessary for the inclusion of all possible events. However, the overall quantity of information can be somewhat overwhelming. Chapter V describes the output methodology, which

consolidated the Path Threads and provided alternative measures of "best" for judging the quality of the consolidated set.

CHAPTER V

OUTPUT METHODOLOGY

Introduction

The algorithm described in Chapter III generates the set of all possible shortest paths, Path Threads. The second objective of this research project was the development of a methodology, which would provide additional meaningful information about the extensive algorithm output.

The path threads that are generated by the algorithm are described by the node*threads, which are traversed in the path. Additionally, the algorithm generates the measure of performance along the arcs, which corresponds to each specific shortest path. The output methodology initially consolidates the path threads by combining the path threads that traverse the same nodes into a single Combo Path. To appropriately describe the Combo Paths, the thread information is combined to generate the measures of performance along the arcs for the Combo Path. A series of sub-procedures, Combine_Path_Threads, Obtain_Combo_Thread_Info, and Find_Arc_Values_On_ Combined_ Path, develop a consolidated solution set of the shortest paths.

Since it is unlikely that there will be only one unique Combo Path, it is important to provide a means for comparing the solutions in the consolidated set. Additionally, different decision-makers may not agree on what defines the "best" shortest path solution. Thus, a sub-objective of the second objective was to develop a methodology that would allow the evaluation of alternative attributes of the consolidated set of solutions. Each attribute of a shortest path solution gives the decision-maker information about the

quality of each path, given a specific objective. The attributes are generally independent of one another and may or may not result in comparable conclusions. A decision-maker may choose one single attribute or combine the qualities of several attributes to determine the "best" among the consolidated set of solutions. Based on the decision-maker's own specific definition of "best", he/she will be better prepared to select the "best" path from among the set of shortest paths.

The consolidated solution set is analyzed and ranked according to three criteria, (1) non-dominance, (2) minimization of regret, and (3) existence of dominant arcs in the sub-procedures Find_NonDominated_Paths, Minimize_Regret, and Find_Path_Points, respectively. Non-dominance and minimization of regret are criteria used in existing literature under conditions of uncertainty, the dominant arc criteria is a criteria created for this research project [4, 18]. Additionally, the set of sub-paths is generated. The sub-paths that are contained among the multiple shortest paths are generated and described in the series of sub-procedures Find_SubPaths, Check_Identical_SubPaths, and Find_Shared_SubPath_ Info.

Combine Path Threads

The path information generated by the algorithm consists of both the nodes traversed and the measurement of performance along the arcs necessary for that path to be considered a shortest path. As shown in Chapter III, the algorithm would generate the following shortest paths for the network in Figure 12.

Path 1 = 1*1 to 3*1 to 4*1

Path 2= 1*1 to 2*1 to 4*2

Path 3 = 1*1 to 3*1 to 4*2

Note that Path 1 and Path 3 traverse the same nodes and are therefore identical paths. The sub-procedure Combine_Path_Threads, shown in Figure 27, searches all shortest paths to find all paths that traverse the same nodes and combines them into Combo Paths. The Combine_Path_Threads sub-procedure yields the information that is easily seen in the trivial example from Figure 12.

Combo1 = Path1 and Path 3 = Node1 to Node 3 to Node 4

Combo2 = Path 2 = Node 1 to Node 2 to Node 4

As the paths are combined, the threads of each member of the ComboPath are collected. The information contained in the threads can be extremely useful when comparing the set of shortest paths. Recall in Figure 14, the idea of cut-points was introduced for Node 4 of the network in Figure 13. The cut points of Node 4 are {4, 5, 7}. The w(k) signify the shortest distance to Node 4. This represents, the shortest path length interval [4, 7] being cut into a series of sub-intervals, [4, 5] and [5, 7]. The output methodology now rejoins these sets where appropriate to find a single w(k) for each ComboPathNumber. The sub-procedure Obtain_Combo_Thread_Info (Appendix G) searches the paths contained in each combo number and collects all threads that exist at each traversed node.

The low thread and high thread are found for each node traversed in the ComboPath. The thread information, consisting of the low thread and high thread, of each node traversed in each ComboPath is collected and combined.

The symbol j is defined to be the lowest thread occurring at node b contained in Combo M, k is the highest thread occurring at node b contained in Combo M, and $w^M(k)$

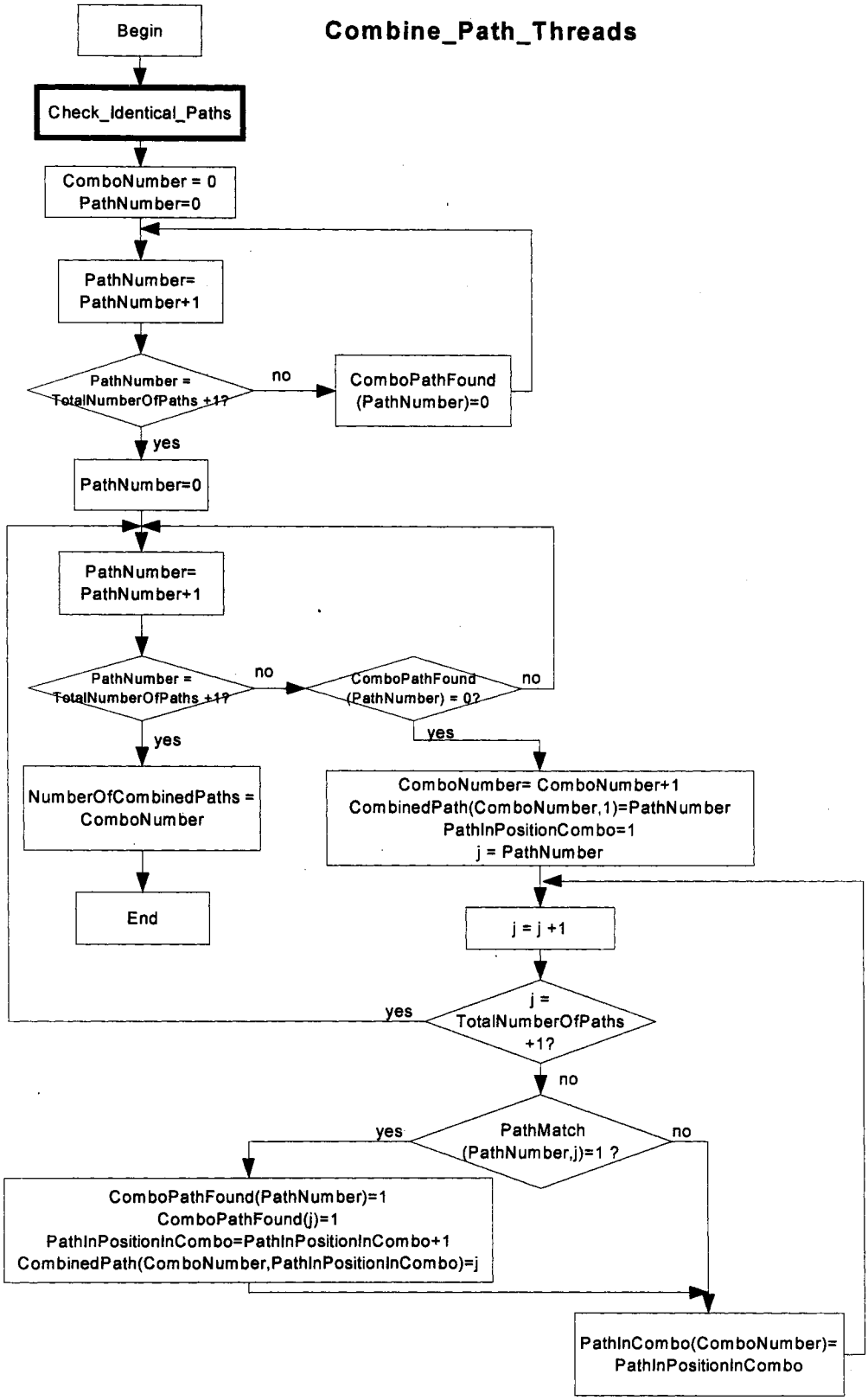


Figure 27: Flow Chart Combine_Path_Threads

is the combined w(k) at node b in Combo M. Given, $w^M_k(b) = [\hat{L}_{bk}, \hat{U}_{bk}]$ and $w^M_j(b) = [\hat{L}_{bj}, \hat{U}_{bj}]$, this implies that $w^M(b) = [\hat{L}_{bk}, \hat{U}_{bj}]$.

For Combo 1, the low and high thread for Node 1 and Node 3 is thread 1. Therefore, the w(k) for Node 1 and Node 3 do not need to be combined. The low thread for Node 4 is 1 and the high thread for Node 4 is 2. These two threads are combined as [4, 5] $\cup$ [5, 7] = [4, 7]. For Combo 1, w(1) equals [0, 0], w(3) equals [2, 4], and w(4) equals [4, 7]. Combo 1 = 1*1 to 3*1 to 4*1 + 1*1 to 3*1 to 4*2= 1*1 to 3*1 to 4*(1+2) = [0, 0] to [2, 4] to [4, 7].

For Combo 2, the low and high thread for Node 1 and Node 2 is thread 1. The low and high thread of Node 4 is 2. Therefore, there is no combination for w(k) in Combo 2. For Combo 2, w(1) equals [0, 0], w(2) equals [1, 2], and w(4) equals [5, 7]. Combo2 = 1*1 to 2*1 to 4*2 = [0, 0] to [1, 2] to [5, 7].

The information regarding combined threads in a shortest path is essential in the methodology to compare the shortest paths. This information is most useful when comparing the combined paths on a basis of non-dominance. Recall that for each path the algorithm generates, the measure of performance along an arc is calculated with respect to the node*threads that the path traversed. The Combo 1 shortest path from Node 1 to Node 3 to Node 4 has a shortest path interval length of [4, 7] units. The Combo 2 shortest path from Node 1 to Node 2 to Node 4 has a shortest path interval length of [5, 7] units.

Specifically, the measure of performance corresponding to arc (j, k) equals w(k) − w(j). However, similar to the non-combined paths, the interval value of the measurement of performance along the path can be counter-intuitive. The sub-procedure

Find_Arc_Values_On_Combined_Path (Figure 25) generates the measure of performance along the arcs with respect to the combined path thread information. The arc(1,2) = [1, 2] and arc(2, 4) = [4, 5] altered measures of performance are based on the interval length of the ComboPath. Combo1 arc(1, 3) = [2, 4] and arc(3, 4) = [2, 3]. The combo path length is weighted along each arc contained on the combo path based on the measure of performance possible along the arcs and path.

Given:

Combo M

EndNode = n

$c(a, b) = [L_{ab}, U_{ab}]$

$w^M(b) = [\hat{L}_{bk}, \hat{U}_{bj}]$; low thread = k, high thread = j

Let:

$c^*(a, b) = [L^*_{ab}, U^*_{ab}]$; the measure of performance along arc (a, b) yielding the shortest path is:

$$L^*_{ab} = \hat{L}_{bk} - \hat{L}_{ak}$$

$$U^*_{ab} = L^*_{ab} + \frac{(U_{ab} - L_{ab})(\hat{U}_{nj} - \hat{L}_{nk})}{\sum_{(a,b)\in P} (U_{ab} - L_{ab})}$$

The Arc Low value is determined by subtracting the terminating low node value from the originating low node value. The portion of the cumulative path span portioned to arc j is based on the original bounds on arc j. The measure of performance along the arc corresponding to each Combo Path is found in the sub-procedure Find_Arc_Values_On_ Combined_Path as seen in Appendix H.

The combined path information specific to the end threads yields all possible shortest path lengths possible for each Combo Path. The cumulative path lengths for a specific path are possible, but not guaranteed. Due to the uncertainty of the interval-network, a truly optimal shortest path may not exist. An optimal path would be a path that would guarantee, regardless of the measures of performance along all the arcs, that the optimal path would always yield the shortest path. The network shown in Figure 11 shows a network with an optimal path, whose cumulative length is [2, 4]. Generally, a truly optimal path will not exist in an interval-network. Although not an optimal path, a non-dominated path is a desirable characteristic of a shortest path of a network.

A non-dominated shortest path is a path in which each its cumulative path length range is contained in the set of all possible shortest path lengths.

Given EndNode = n, cardinality of w(n) = c, $w_1(n) = [\hat{L}_{n1}, \hat{U}_{n1}]$, $w_c(n) = [\hat{L}_{nc}, \hat{U}_{nc}]$, Combo M Lower Bound = $\sum_{(a,b)\in M} L_{ab}$,and Combo M Upper Bound = $\sum_{(a,b)\in M} U_{ab}$. Combo M is a non-dominated path if $\sum_{(a,b)\in P} L_{ab} = \hat{L}_{n1}$ = Short Lower Bound, and $\sum_{(a,b)\in P} U_{ab} = \hat{U}_{nc}$ = Short Upper Bound. The algorithm searches through the ComboPaths and EndThreads. If the Combo Path satisfies both of these conditions it is a non-dominated path. Figure 28 shows the flow chart for sub-procedure Find_NonDominated_Paths. Table XVIII shows the Combo Path non-dominance information including short and path upper and lower bounds for the network shown in Figure 12.

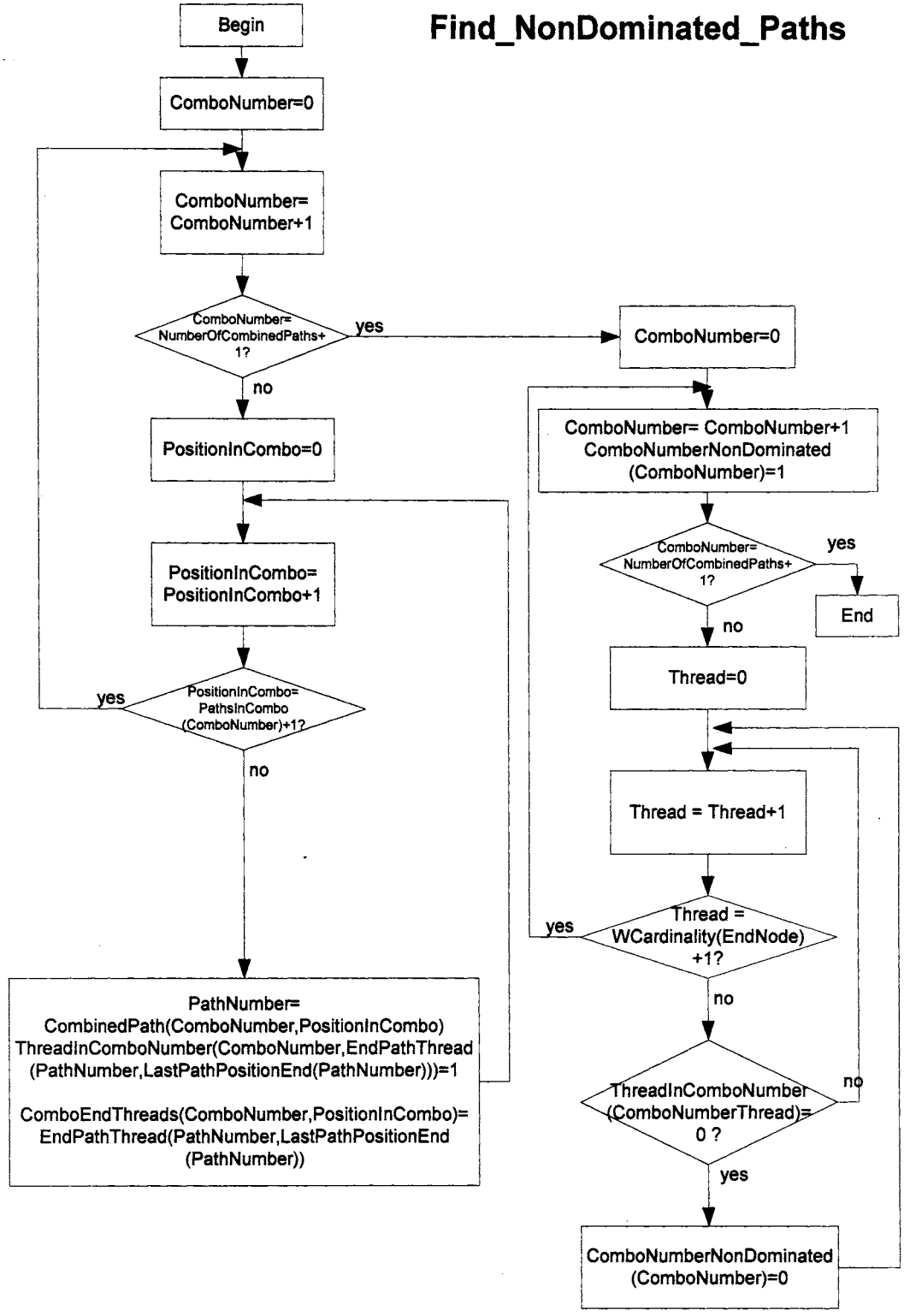


Figure 28: Flow Chart Find_NonDominated_Paths

It is possible that a network may contain no or more than one non-dominated path. The primary significance of a non-dominated path is that it is a path with the shortest possible cumulative length and the path that cannot be worse than the longest shortest path. Recall, that an interval-network does not provide any information about the probability of any specific quantity for the measure of performance along an arc. Therefore, it is unknown how likely a path is to yield any specific cumulative length. A decision-maker that chooses a non-dominated path as the "best" path is willing to take any risk for the possibility (regardless of how small) of achieving the goal of shortest possible path length.

Table XVIII
Non-Dominance Information for Figure 12 Network

	Short Lower Bound	Short Upper Bound	
	4	7	
	Combo Lower Bound	Combo Upper Bound	Non-Dominated Path (Y/N)
Combo 1	4	7	Y
Combo 2	5	7	N

The non-dominance information shows all possible End Thread lengths possible for any ComboPath, including the smallest possible shortest path length for each Combo Path. The sub-procedure Minimize_Regret, shown in Figure 29, further reduces this information to give the user an overview of the shortest possible shortest path length and the longest possible shortest path length for each Combo Path.

A decision-maker that bases his/her choice for "best" on non-dominance disregards risk. However, some decision-makers may want to base their choice of "best" on a desire to avoid risk, or minimize regret. This type of decision-maker is concerned primarily

with protecting against the worst-case for each Combo Path, or more specifically, with knowing the difference between the worst-case and the best-case scenarios.

In the sub-procedure Combine_Path_Threads, the Combo Paths are sequentially numbered according to their shortest possible cumulative path length. That is, in the best case for each Combo Path, Combo P would have a shorter possible cumulative path length than Combo P+1. The possible shortest path lengths corresponding to each Combo Path is provided in the non-dominance information. However, this information only corresponds to the possible shortest path. Some Combo Paths contain possible cumulative lengths that are greater than the longest possible shortest path. The longest possible shortest path is the upper bound for the last End Thread. The Combo Path information for the cumulative length that may not be a shortest path, is important information to a decision-maker that is concerned with the minimization of regret.

The minimize regret information addresses the issue of worst case for each Combo Path. Each Combo Path contains a span of possible path lengths, if that path were actually a shortest path. Whether the path is the true shortest path is unknown. Additionally, each Combo Path has a span of path possible lengths if that path were implemented.

There are two types of Regret that a decision-maker may want to consider. If a Combo Path were chosen and it wasn't actually the shortest path, that would be considered "regret". Moreover, if a Combo Path were implemented, the user would be regretful if the measure of performance along each arc in the Combo Path were at its upper bound.

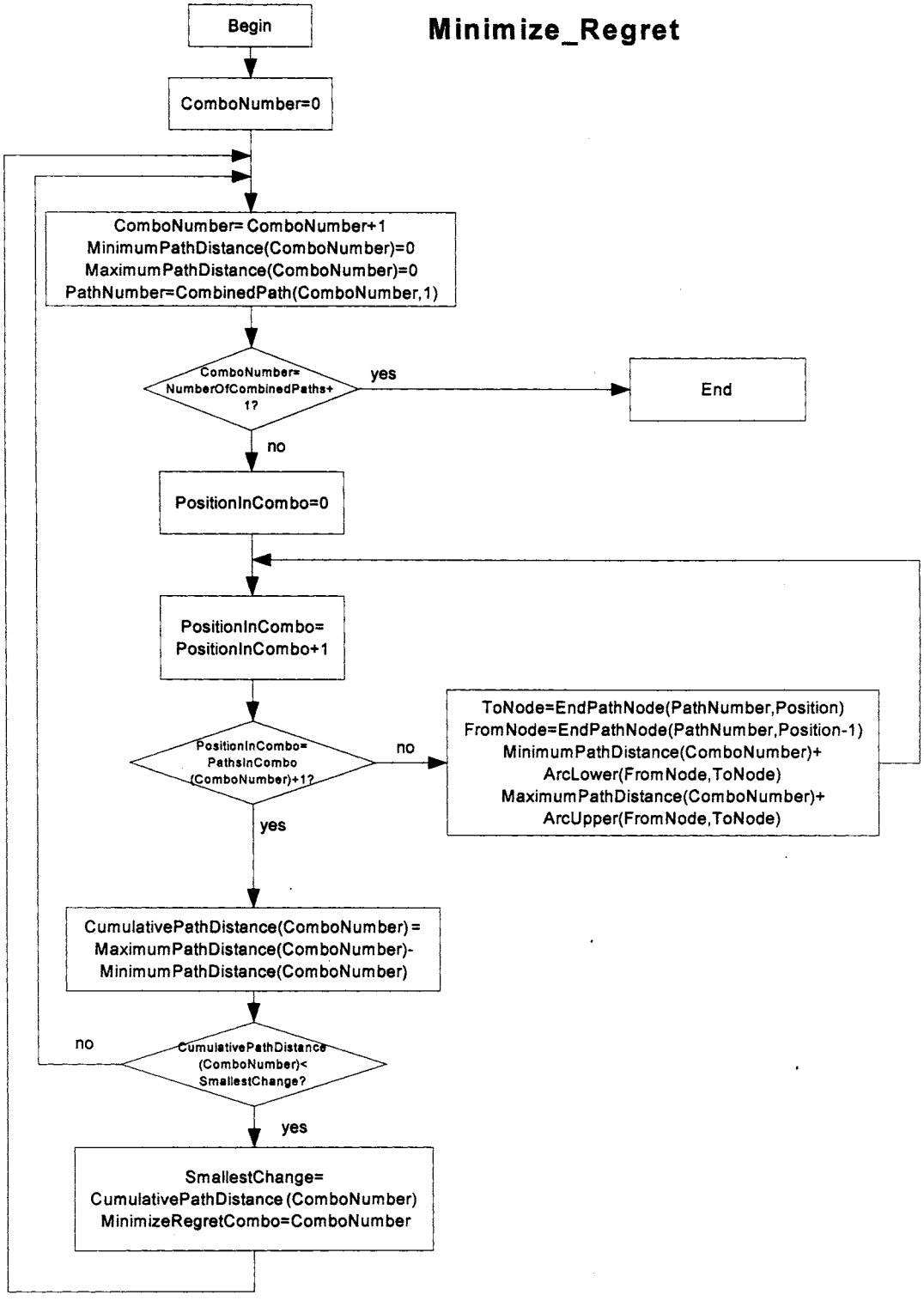


Figure 29: Flow Chart Minimize_Regret

Regret A depends only on Combo Path P. Regret results when the decision has been made to implement Combo Path P and the length of Combo Path P is the longest length possible rather than the shortest length possible. Regret A for a Combo Path P is defined as the longest possible length of Combo Path P minus the shortest possible length for a Combo Path P.

Regret B depends on both Combo Path P and a non-dominated path with the shortest possible length. The shortest possible path length is the Lower Bound of End Node Thread 1. The path(s) that contains the shortest possible length is a non-dominated path. Regret B results when Combo Path P is implemented, but the decision maker is concerned with what may have happened if a non-dominated path had been chosen and the non-dominated path had been at its shortest possible value. Regret B for Combo Path P is the difference between the longest possible path length of Combo Path P and the overall shortest possible path length.

Path Arc Points Attribute and Shared Sub-Paths

The non-dominance and minimize regret information are based on resulting cumulative path lengths for the Combo Paths. However, the algorithm has generated all possible shortest paths and the solution set itself contains information about the network. Specifically, the shortest paths consist of nodes and arcs. Since, the primary objective is the minimization of the measure of performance along the arcs, the arc information in the solution set is also important. Many of the arcs in the algorithm consistently appear as a part of a shortest path. The information that consistently appears in the set of shortest paths could be as useful as the shortest paths themselves. The arcs that are contained in

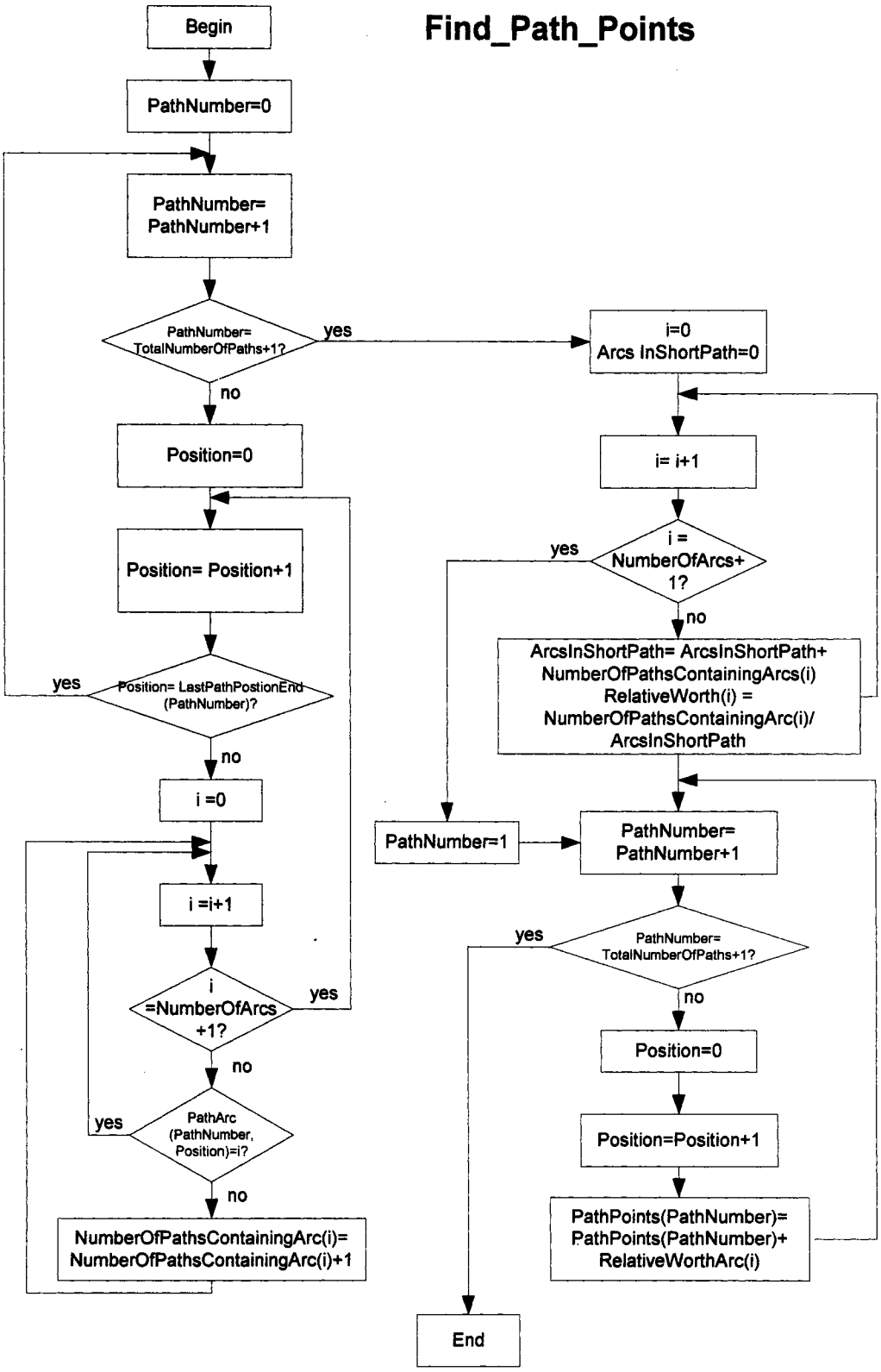


Figure 30: Flow Chart Find_Path_Points

more than one Combo Path are considered dominant arcs. An arc's dominance in the solution set is based on the number of occurrences in the set of Combo Paths.

The sub-procedure Find_Path_Points, shown in Figure 30, calculates Path Arc Points for each Combo Path. Each Combo Path is given Path Arc Points relative to the ranking of each arc that appears in the Combo Path. Each shortest path is given a ranking of Path Arc Points. The relative worth of each arc is based on the number of times that the arc appears in the shortest path set. The Path Arc Points are a sum of relative worth of each arc in the Combo Path divided by the number of arcs in the path. This calculation gives the user a per arc worth of the path.

A Combo Path with high Path Arc Points contains arcs that are "often" contained in the true shortest path. Recall that we are unaware of the probability associated with each Combo Path. The Path Arc Points is simply an enumeration technique. The Path Arc Points are important information when the decision-maker is interested in the shortest path in a network that will be traversed many times and the long-run conditions are important.

Below is the procedure for finding the Path Arc Points.

Given:

$G = (N, A, c)$, i.e., G is acyclic and directed.

$N = \{1, 2, ..., n\}$

$A = \{(a, b) \mid (a, b) \in A \rightarrow a < b\} = \alpha_i, \ \{\alpha_i \mid i = 1, 2, ..., m\}$

$CP = \text{Combo Paths} = \{P_i \mid P_i(1, ..., a, b, ..., n)\}$

$\text{Relative worth } \alpha_i = \sum_{P_i \in CP} \alpha_i \in P_i$

$$\text{Path Arc Points} = \sum_{\alpha_i \in P_i} \text{Relative Worth } \alpha_i \text{ /Number of arcs in the Combo Path}$$

As some arcs are dominant throughout the set of shortest paths, there are also sub-paths which appear throughout the set of shortest paths. All sub-paths of each shortest path are identified by the length of the sub-path. First, the sub-paths of length three are found. Finally, the sub paths of the length of the longest shortest path are found. To more easily implement the Find_Path_Points sub-procedure, the arcs in the network are numbered in the sub-procedure Get_Arcs. The Get_Arcs sub-procedure is shown in the Appendix K. The sub-procedure Find_SubPaths, shown in Figure 31, generates all sub-paths. Once the sub-paths have been generated, matching sub-paths are noted in the sub-procedure, Check_Identical_SubPaths (Appendix I). The Combo Paths that contain the sub paths are written to a Microsoft Excel [10] spreadsheet by the sub-procedure Find_Shared_SubPath_Info (Appendix J).

The shared sub-path information is important because of the possible dependence among Combo Paths. If two Combo Paths have a shared sub-path, then the cumulative measure of one path is directly related to the cumulative measure of performance of the other path.

The shared sub-path information is easily acquired after the algorithm has been implemented and is intended to give the decision-maker additional information about the network.

Output Methodology for 10-Node Network III

The algorithm generated three shortest paths for the 10-Node shown in Figure 23. The sub-procedures Combine_Path_Threads, Obtain_Combo_Thread_Information, and

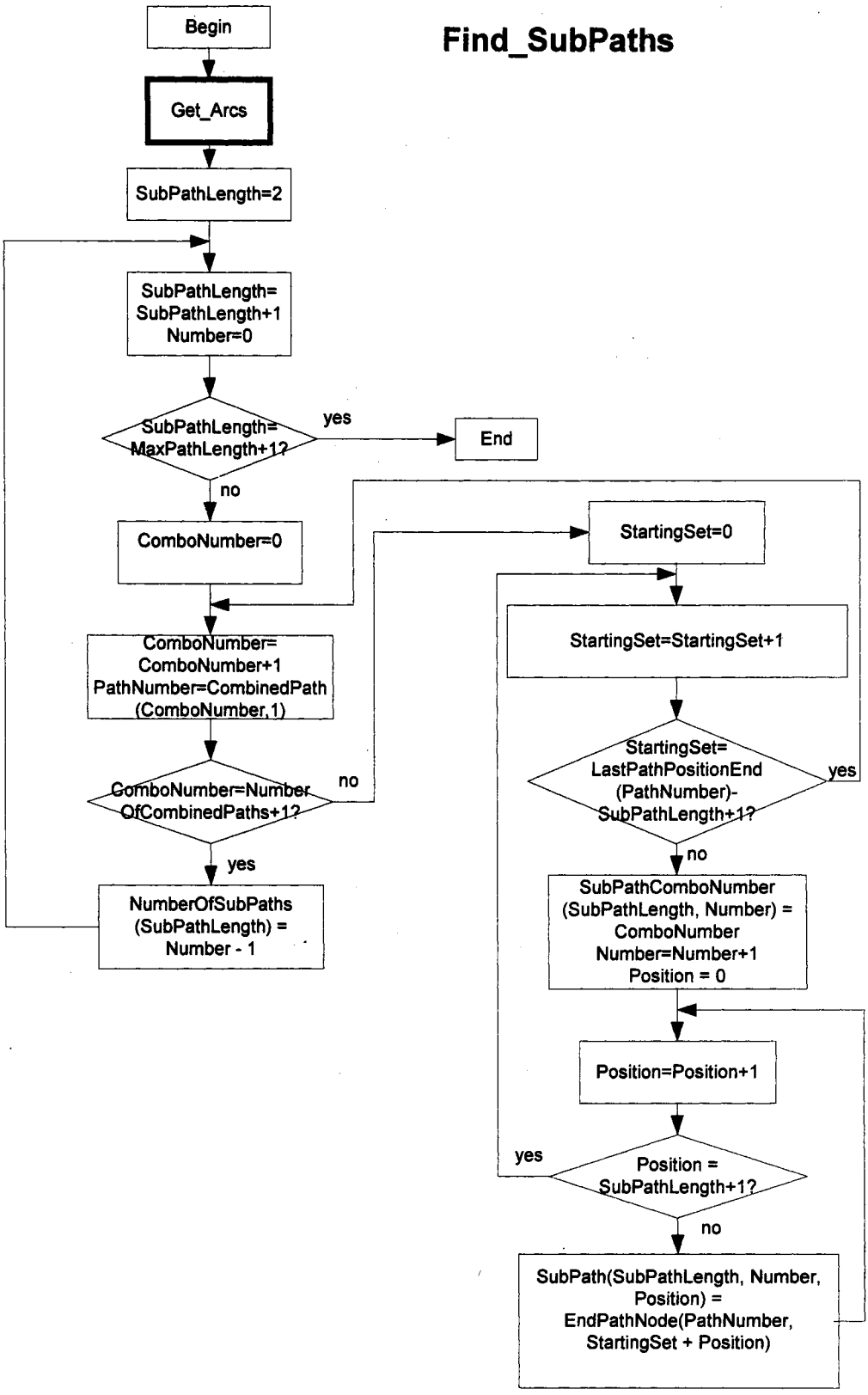


Figure 31: Flow Chart Find_SubPaths

Find_Arc_Values_On_Combined_Path combined these three shortest paths into two ComboPaths. These paths are shown in Table XIX. For the network in Figure 23, there are only two possible shortest paths. The measures of performance along each arc are displayed in Table XIX in the Arc Low and Arc High columns.

Table XIX
Combined Paths for 10-Node Network III

Arc	To	From	Lower Bound	Upper Bound	Low Thread	High Thread	Arc Low	Arc High
Arc1	1	3	21	25	1	1	21	25
Arc2	3	6	42	46	1	1	42	46
Arc3	6	10	16	20	1	2	16	20
Total							79	91
Arc1	1	4	42	46	1	1	42	42.75
Arc2	4	7	15	19	1	1	15	15.75
Arc3	7	8	13	17	1	1	13	13.75
Arc4	8	10	18	22	2	2	18	18.75
Total							88	91

Recall that the algorithm generates the shortest paths for every possible measure of performance for each of the arcs in the interval-network. Figures 21 and 22 display the traditional network where the measures of performance are the lower bounds and upper bounds of Figure 23 respectively. The networks in Figures 21 and 22 contain the same shortest path (Node 1 → Node 3 → Node 6 → Node 10). Specifically, for the network in Figure 23, Combo 2 (Node 1 → Node 4 → Node 7 → Node 8 → Node 10) would not have been part of the solution set had a traditional algorithm been implemented with constant measures of performance at the lower and upper bounds.

The non-dominance information shown in Table XX provides an overview of the possible shortest path distances. Combo 1 is the non-dominated shortest path for the network in Figure 23. However, Combo 1 is not guaranteed to be the shortest path. For instance, Combo 2 could have a cumulative length of 88.5 and Combo 1 could have a

cumulative length of 91. Additionally, as Table XXI shows, if Combo 1 were chosen, it could only deviate 12 units from the best case shortest path of 79 units.

Table XX
Non-Dominance Information for 10-Node Network III

	Short Lower Bound	Short Upper Bound	
	79	91	
	Combo Lower Bound	Combo Upper Bound	Non-Dominated Path (Y/N)
Combo 1	79	91	Y
Combo 2	88	104	N

Table XXI
Regret for 10-Node Network III

Combo	Path Lower Bound	Path Upper Bound	Regret A	Regret B [UB−79]
Combo 1	79	91	12	12
Combo 2	88	104	16	25

The shortest path solution set consists of 7 different arcs. No arc was a member of the shortest path solution set more than once. Therefore, each arc had the same relative worth, 1 and each path had the same number of Path Arc Points, namely, 1. Additionally, there were no sub-paths that existed in more than one of the shortest paths in the solution set. Combo 1 would be considered the "best" path regardless of which attribute, non-dominance, minimize regret A, minimize regret B or Path Arc Points, was used by a decision maker.

Output Methodology 10-Node Network IV

The algorithm generated 538 shortest paths for the 10-Node network shown in Figure 24. The algorithm methodology combined these 538 shortest paths into 14 Combo Paths.

The Combo Path information shown in Table XXII is a considerable condensation from the amount of information that the algorithm initially generated.

Table XXII
Combined Paths for 10-Node Network IV

Combo	Arc	To	From	Low Thread	High Thread	Arc Low	Arc High		Combo	Arc	To	From	Low Thread	High Thread	Arc Low	Arc High
1	1	1	3	1	1	21	30		8	1	1	2	1	1	28	31.4
	2	3	6	1	3	42	60			2	2	5	1	1	21	23.6
	3	6	10	1	20	16	20			3	5	7	2	5	14	15.7
										4	7	10	9	20	35	39.3
2	1	1	4	1	2	42	54.4									
	2	4	7	1	6	15	18.4		9	1	1	3	1	1	21	23.4
	3	7	8	1	9	13	17.8			2	3	4	2	2	28	31.2
	4	8	10	2	20	18	19.4			3	4	7	3	5	15	16.3
										4	7	10	10	20	35	39
3	1	1	4	1	2	42	50.5									
	2	4	7	1	5	15	17.4		10	1	1	3	1	1	21	23.3
	3	7	10	3	20	35	42.1			2	3	4	2	2	28	31
										3	4	6	3	3	35	38.8
4	1	1	4	1	2	42	50.3			4	6	10	11	20	16	17
	2	4	6	2	3	35	41.9									
	3	6	10	4	20	16	17.8		11	1	1	3	1	1	21	23.1
										2	3	6	1	1	42	46.2
5	1	1	2	1	1	28	33.3			3	6	7	5	6	7	7.69
	2	2	5	1	2	21	25			4	7	8	5	9	13	14.6
	3	5	7	2	6	14	16.7			5	8	10	12	20	18	18.5
	4	7	8	2	9	13	16.1									
	5	8	10	5	20	18	18.9		12	1	1	3	1	1	21	22
										2	3	6	1	1	42	44
6	1	1	2	1	1	28	33.1			3	6	7	5	5	7	7.33
	2	2	5	1	2	21	24.9			4	7	10	16	20	35	36.7
	3	5	8	3	9	28	33.1									
	4	8	10	6	20	18	18.9		13	1	1	4	1	1	42	44.1
										2	4	5	2	2	19	19.1
7	1	1	3	1	1	21	24.9			3	5	7	6	6	14	14.7
	2	3	4	2	2	28	33.1			4	7	8	7	9	13	13.8
	3	4	7	3	6	15	17.1			5	8	10	17	20	18	18.2
	4	7	8	3	9	13	16									
	5	8	10	6	20	18	18.9		14	1	1	4	1	1	42	43.6
										2	4	5	2	2	19	19.1
										3	5	8	8	9	28	29.1
										4	8	10	18	20	18	18.2

The non-dominance information shown in Table XXIII provides an overview of the possible shortest path distances. Combo 1 is the non-dominated shortest path for the

network in Figure 24. However, Combo 1 is not guaranteed to be the shortest path. Additionally, as Table XXIV shows, Combo 1 has the minimum regret of types A and B. Combo 1 could only deviate 31 units from the best-case shortest path of 79 units.

Table XXIII:
Non-Dominance Information for 10-Node Network II

	Short Lower Bound	Short Upper Bound	
	79	110	
	Combo Lower Bound	Combo Upper Bound	Non-Dominated Path (Y/N)
Combo 1	79	110	Y
Combo 2	88	120	N
Combo 3	92	130	N
Combo 4	93	130	N
Combo 5	94	130	N
Combo 6	95	130	N
Combo 7	95	130	N
Combo 8	98	140	N
Combo 9	99	140	N
Combo 10	101	140	N
Combo 11	101	140	N
Combo 12	105	150	N
Combo 13	106	140	N
Combo 14	107	140	N

Therefore, Combo 1 would be the "best" solution considering the non-dominance and minimize regret attributes. However, as displayed in Table XXV, Combo 2 is the shortest path with the largest Path Arc Points. A path with high Path Arc Points defines a Combo Path whose arcs consistently lie on a shortest path.

The sub-paths of the network in Figure 24 are shown in Table XXVI. The sub path, Node 7 → Node 8 → Node 10, is contained in 5 of the 14 Combo Paths. This information adds to the benefit of Combo 2 as a good solution for the shortest path. The output methodology is intended to give the decision-maker additional information in order to make an informed decision. The decision-maker would have to weigh the benefits of each Combo Path according to his/her specific needs.

Table XXIV
Regret for 10-Node Network IV

Combo	Path Lower Bound	Path Upper Bound	Regret A	Regret B [UB−79]
Combo 1	79	110	*31	*31
Combo 2	88	120	32	41
Combo 3	92	130	38	51
Combo 4	93	130	37	51
Combo 5	94	130	36	51
Combo 6	95	130	35	51
Combo 7	95	130	35	51
Combo 8	98	140	42	61
Combo 9	99	140	41	61
Combo 10	101	140	40	61
Combo 11	101	140	39	61
Combo 12	105	150	45	71
Combo 13	106	140	34	61
Combo 14	107	140	33	61

Table XXV
Path Points for 10-Node Network IV

Combo Number	Path Arc Points	Combo Number	Path Arc Points
1	4.00	8	3.25
*2	*5.25	9	4.25
3	4.33	10	3.50
4	3.33	11	4.60
5	4.20	12	3.75
6	3.75	13	4.40
7	5.00	14	4.00

Table XXVI
Sub Paths for 10-Node Network IV

Sub Path					Combo Number Which Contains Sub Path				
4	7	8			2	7			
7	8	10			2	5	7	11	13
4	7	10			3	9			
4	6	10			4	10			
2	5	7			5	8			
5	7	8			5	13			
5	8	10			6	14			
3	4	7			7	9			
3	6	7			11	12			
4	7	8	10		2	7			
5	7	8	10		5	13			

Output Methodology for 7 Node Network

The algorithm generated 14 shortest paths for the 7-Node network shown in Figure 24. The output methodology combined these 14 shortest paths into 5 Combo Paths. These paths are shown in Table XXVII. The non-dominance information shown in Table XXVIII provides an overview of the possible shortest path distances. Note their are no non-dominated shortest paths for the network in Figure 25. Since there are no non-dominated paths, a decision-maker that uses alternate attributes of the "best" shortest path, as Regret A, Regret B or Path Arc Points.

Table XXVII:
Combined Paths for 7-Node Network

Combo	Arc	To	From	Low Thread	High Thread	Arc Low	Arc High
1	1	1	3	1	1	4	6.67
	2	3	5	1	2	5	6.78
	3	5	7	1	3	6	9.56
2	1	1	2	1	1	3	4.45
	2	2	3	1	1	1	3.18
	3	3	5	1	2	5	6.45
	4	5	7	1	3	6	8.91
3	1	1	2	1	1	3	4.4
	2	2	5	2	2	7	9.8
	3	5	7	2	3	6	8.8
4	1	1	3	1	1	4	7
	2	3	6	1	1	4	5
	3	6	7	2	3	8	11
5	1	1	2	1	1	3	4.56
	2	2	3	1	1	1	3.33
	3	3	6	1	1	4	4.78
	4	6	7	2	3	8	10.3

Additionally, as Table XXIX shows, Combo 4 is the shortest path with a minimum Regret A of 7 units and Regret B of 8 units. As seen in Table XXX, Combo 2 is the shortest path with the largest Path Arc Points. There are two sub-paths of the network in

Figure 25. Sub-Path 1, Node 3 → Node 5 → Node 7, is contained in Combo 1 and Combo 2. Sub-Path 2, Node 3 → Node 6 → Node 7, is contained in Combo 4 and Combo 5.

Table XXVIII:
Non-Dominance Information for 7-Node Network

	Short Lower Bound	Short Upper Bound	
	15	23	
	Combo Lower Bound	Combo Upper Bound	Non-Dominated Path (Y/N)
Combo1	15	24	N
Combo2	15	26	N
Combo3	16	26	N
Combo4	16	23	N
Combo5	16	25	N

For a decision-maker who is attempting to "break the tie" between Combo 1 and Combo 2, Combo 1 contains a stronger Regret A and Regret B attribute. However, Combo 2 has a stronger Path Arc Points attribute. A decision-maker should have a clear understanding of his/her objectives in order to choose the "best" path from among the competing Combo Paths.

Table XXIX
Regret for 7-Node Network

Combo	Path Lower Bound	Path Upper Bound	Regret A	Regret B [UB-15]
Combo1	15	24	9	9
Combo2	15	26	11	11
Combo3	16	26	10	11
Combo4	16	23	*7	*8
Combo5	16	25	9	10

Table XXX
Path Points for 7-Node Network

Combo Number	Path Arc Points
1	2.33
2	2.50
3	2.33
4	2.00
5	2.25

Output Methodology for 6-Node Network

The network in Figure 26 is identical in structure to the network that Okada and Soper used as an example in their shortest path analysis [11]. The network introduced by Okada and Soper contained trapezoidal fuzzy numbers as the measure of performance along the arcs. The Avery Shortest Path algorithm generated 80 shortest paths for the 6-Node Network shown in Figure 26. The output methodology combined these 80 shortest paths into 5 Combo Paths. This set of all possible shortest paths is shown in Table XXXI. There are no non-dominated paths for the network in Figure 26. The non-dominance information shown in Table XXXII provides an overview of the possible shortest path distances.

Table XXXI
Combined Paths for 6-Node Network

Combo	Arc	To	From	Low Thread	High Thread	Arc Low	Arc High
1	1	1	2	1	1	10	28.78
	2	2	3	1	2	35	44.39
	3	3	5	1	3	8	9.88
	4	5	6	1	10	50	96.95
2	1	1	3	2	2	52	70
	2	3	5	2	3	8	10
	3	5	6	2	10	50	100
3	1	1	2	1	1	10	26.39
	2	2	5	3	3	52	62.65
	3	5	6	3	10	50	90.96
4	1	1	2	1	1	10	26.42
	2	2	3	1	2	35	43.21
	3	3	4	1	3	10	18.21
	4	4	6	4	10	70	92.16
5	1	1	3	2	2	52	67.71
	2	3	4	2	3	10	18.73
	3	4	6	5	10	70	93.56

Table XXXII
Non-Dominance Information for 6-Node Network

	Short Lower Bound	Short Upper Bound	
	103	180	
	Combo Lower Bound	Combo Upper Bound	Non-Dominated Path (Y/N)
Combo1	103	185	N
Combo2	110	180	N
Combo3	112	195	N
Combo4	125	192	N
Combo5	132	187	N

Additionally, as Table XXXIII shows, Combo 5 is the shortest path with the minimum Regret A of 55 units and Combo 2 has minimum regret B of 73 units. As seen in Table XXXIV, Combo 1 is the shortest path with the largest Path Arc Points. There are two sub-paths of the network in Figure 26. Sub-Path 1, Node 3 → Node 5 → Node 6, is contained in Combo 1 and Combo 2. Sub-Path 2, Node 3 → Node 4 → Node 6, is contained in Combo 4 and Combo 5.

Table XXXIII
Regret for 6-Node Network

Combo	Path Lower Bound	Path Upper Bound	Regret A	Regret B [UB−103]
Combo1	103	185	82	82
Combo2	110	180	70	*73
Combo3	112	195	83	92
Combo4	125	192	67	89
Combo5	132	187	*55	84

Table XXXIV:
Path Points for 6-Node Network

Combo Number	Path Arc Points
1	2.50
2	2.33
3	2.33
4	2.25
5	2.00

As previously mentioned, Okada and Soper determined the shortest path of a network with trapezoidal fuzzy numbers. For proper comparison, Table XXXV shows the trapezoidal node values identified by Okada and Soper. Each trapezoidal number (Left Top, Right Top, Left Bottom, Right Bottom) was converted to an interval [Lower Bound = Left Top – Left Bottom, Upper Bound = Right Top + Right Bottom]. The trapezoidal numbers are listed under the column heading Node'T', e.g., "2T". The interval numbers are under the column heading Node'I', e.g., "2I".

Table XXXV:
Okada and Soper Node Labels [11, p.136]

Node	2T	2I	3T	3I	4T	4I	5T	5I	6T	6I
Left Top	20	10	62	52	71	55	75	62	137	103
Right Top	20	30	65	70	77	95	80	95	149	185
Left Bottom	10		10		16		13		34	
Right Bottom	10		5		18		15		36	
Left Top			58	45	75	62	67	53	146	125
Right Top			60	75	82	90	69	85	162	193
Left Bottom			13		13		14		21	
Right Bottom			15		8		16		30	
Left Top							71	60	141	110
Right Top							74	80	154	180
Left Bottom							11		31	
Right Bottom							6		26	
Left Top									150	132
Right Top									167	187
Left Bottom									18	
Right Bottom									20	

Table XXXVI shows the translated values generated by Okada and Soper under the column heading Node'OS', e.g., "2OS". The node threads generated by the algorithm are under the column heading Node'A', e.g., "2A". The comparable values in Table XXXVII are nearly identical. The lower bound values are identical with one exception. However, the upper bounds of the two sets of node values do not match. The table values corresponding to Okada and Soper's trapezoidal results are nearly identical to the ones generated by the algorithm. The Avery Shortest Path algorithm has given all Node

Threads as a set of nearly disjoint sets. Okada and Soper have combined their node values in a similar manner to the algorithm's results for the End Node.

The node values shown by Okada and Soper are nearly identical to the Combo Path values seen in Table XXXIII. The results of Okada and Soper did not include Combo 3 with path length [112, 195]. Since, Okada and Soper did not show a clear methodology for the creation of their solution set, the reason for this discrepancy is unknown.

Table XXXVI:
Algorithm Node Threads/ Okada and Soper Node Labels

Node		2A	2OS	3A	3OS	4A	4OS	5A	5OS	6A	6OS
Lower	1	10	10	45	45	55	55	53	53	103	103
Upper	1	30	30	52	75	62	95	60	85	110	185
Lower	2			52	52	62	62	60	60	110	110
Upper	2			70	70	72	90	62	80	112	180
Lower	3					72		62	62	112	
Upper	3					90		80	95	125	
Lower	4									125	125
Upper	4									132	193
Lower	5									132	132
Upper	5									142	187
Lower	6									142	
Upper	6									159	
Lower	7									159	
Upper	7									160	
Lower	8									160	
Upper	8									162	
Lower	9									162	
Upper	8									169	
Lower	10									169	
Upper	10									180	

However, Combo 3, Node 1 → Node 2 → Node 5 → Node 6, is a possible shortest path for the interval-network in Figure 26. Table XXXII, containing Non-Dominance Information, shows the circumstances under which Combo 3 will be the shortest path. Additionally, Combo 3 has no shared sub-paths with any other Combo. Therefore, the length of Combo 3 is independent of the length of the other Combo Paths. That is,

Combo 3 could be at its smallest possible length, 112 units, while the other Combo Paths are at their largest possible length.

The output methodology described in this chapter was developed to create a meaningful shortest path solution set without thread information. As multiple shortest paths were often contained in the solution set, various attributes of the shortest paths, Non-Dominance, Regret Type A, Regret Type B, and Path Arc Points were defined to describe alternative qualities of each shortest path. Different attributes were necessary since each decision-maker has his/her own objectives in choosing among the various shortest paths. The selected attributes were independent of one another and did not necessarily result in comparable conclusions. A decision-maker may choose one attribute or he/she may combine two or more of the attributes to define the "best" among the shortest paths. Additionally, sub-path information was introduced to give the decision-maker essential information about the network and therefore the system being modeled.

CHAPTER VI

SUMMARY AND RECOMMENDATIONS

Networks have been used to model science, engineering, and business applications of transportation, communication, mechanical, hydraulic, electrical and economic systems [15]. Traditional network applications have been based on constant-valued arc measures. However, this assumption is often unrealistic and this problem has not been solved with arc values that are contained in some known interval.

The primary objective of this research was the development of an algorithm for the interval-valued problem that would ensure that all possible shortest paths have been generated. The techniques of Qualitative Discrete Event Simulation (QDES) were used to complete this task and a thread generation technique was designed in the algorithm to guarantee this result. The number of iterations required to complete the algorithm was exponential with respect to the number of nodes in the network and the overall speed and efficiency of the algorithm was not a priority.

The traditional problem is apt to have a unique shortest path, but it is likely that there will be multiple solutions for the interval-valued problem. Therefore, a second objective of this research was the development of a methodology that would provide for an intelligent consolidation of the initial set of solutions. This objective was also accomplished.

However, since it is unlikely that the reduction in the set of solutions would result in a unique path, it was useful to provide a comparison of the resulting solutions in the consolidated solution set. Additionally, different decision-makers may not agree of what

defines the "best" shortest path solution. Thus, a sub-objective of the second objective was to develop a methodology that would allow the evaluation of alternative attributes of the consolidated set of solutions. Each attribute of a shortest path solution provided the decision-maker with information about the quality of each path, given a specific objective. A decision-maker could choose one single attribute or combine the qualities of several attributes to determine the "best" among the consolidated set of solutions. Based on the decision-maker's own specific definition of "best", he/she would be able to select the "best" path from among the set of shortest paths. Evaluations of three attributes for each shortest path were developed in the output methodology: Non-dominance, Minimize Regret, and Path Arc Points. The attributes were independent of one another and may or may not result in the same conclusion. Each of these shortest path attributes gave the decision-maker information with regard to the quality of each path, given a specific objective.

The algorithm, Avery Shortest Path Algorithm, was tested by implementing a variety of networks. Since the shortest path solution of networks with constant-valued measures of performance was readily available, this type of network was initially solved by the algorithm to verify the solution. Additionally, networks with specific characteristics were solved by the algorithm, e.g., a small solution set of shortest paths. Other networks that contained specific output characteristics, e.g., multiple non-dominated paths and shortest paths with the same "best" solution using all three solution attributes, were evaluated by the algorithm. However, since there is no existing shortest path algorithm for an interval-network, complete verification of the results using existing techniques was not possible. Okada and Soper have given results of a network with fuzzy arc lengths.

The fuzzy arc lengths in the Okada and Soper example were translated into interval-values. The Avery Shortest Path Algorithm had near identical results to those obtained by Okada and Soper [13].

Recommendations

Since the number of iterations required to complete the algorithm was exponential with respect to the number of nodes in the network, the time required to obtain the set of solutions can be a significant problem in a large interval-network. A suggested area of future research would be to determine if improvements could be made in the algorithm to reduce the number of iterations required to generate the complete set of shortest paths.

The shortest path solution set contains path information specific to the threads generated by the algorithm. A second suggested area of future research would be to develop an extension of the algorithm that would combine the threads generated by the existing algorithm before the shortest paths are found.

A third suggested area of future research would be to define additional attributes and develop the methodology for the evaluation of those attributes that could be used to assess the quality of each member of the shortest path solution set.

REFERENCES

[1] Allen, J. F., "Maintaining Knowledge about Temporal Intervals," *Communications of the ACM*, vol. 26, pp. 832-43, 1983.

[2] Cellier, F. E., "Qualitative Modeling and Simulation: Promise or Illusion," *Proceedings of the 1991 Winter Simulation Conference*, 1991.

[3] Chabini, L. and Lan, S., "Adaptations of the A* Algorithm for the Computation of Fastest Paths in Deterministic Discrete-Time Dynamic Networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 3, pp. 60-74, 2002.

[4] Chen, B. and Lin, C.-S., "MinMax Regret Robust 1-Median Location on a Tree," *Networks*, vol. 31. New York: John Wiley & Sons, pp. 93-103, 1998.

[5] Fouche, P. and Kuipers, B. J., "Reasoning About Energy in Qualitative Simulation," *IEEE Transactions of Systems, Man, and Cybernetics*, vol. 22, p. 47, 1992.

[6] Hillier, F. S. and Lieberman, G. J., *Introduction To Operations Research, Seventh Edition*, New York: McGraw-Hill, Inc., 2001.

[7] Ingalls, R. G., "Qualitative Simulation Graph Methodology and Implementation," Ph.D. Dissertation in *Management Science*. Austin, TX: University of Texas, May 1999.

[8] Kelton, W. D., Sadowski, R. P., and Sadowski, D. A., *Simulation with Arena*, New York: McGraw Hill, Inc., 1991.

[9] Law, A. M. and Kelton, W. D., *Simulation Modeling and Analysis, Second Edition*, New York: McGraw Hill, Inc., 1991.

[10] Microsoft, "Microsoft Excel 2002," SP-2 Edition, Redmond, WA: Microsoft Corporation, 2001.

[11] Microsoft, "Microsoft Visual Basic," 6.3.8863 Edition, Redmond, WA: Microsoft Corporation, 2001.

[12] Morris, W., *The American Heritage Dictionary of the English Language, New College Edition*, Boston, MA: Houghton Mifflin Company, 1979.

[13] Okada, S. and Soper, T., "A Shortest Path Problem on a Network with Fuzzy Arc Lengths," *Fuzzy Sets and Systems*, vol. 109, pp. 129-140, 2000.

[14] Rardin, R. L., *Optimization in Operations Research*, Upper Saddle River, NJ: Prentice Hall, Inc., 1998.

[15] Rockefeller, R. T., *Network Flows and Monotropic Optimization*, New York: John Wiley & Sons, Inc., 1984.

[16] Savage, E. L. and Schruben, L. W., "Eliminating Event Cancellation in Discrete Event Simulation," *Proceedings of the 1995 Winter Simulation Conference*, 1995.

[17] Schruben, L. W. and Yucansan, E., "Simulation Graphs," *Proceedings of the 1988 Winter Simulation Conference*, 1988.

[18] Sudharsanan, S. R., "Fuzzy Distance Approach to Routing Algorithms for Optimal Web Path Estimation," *Proceedings of the 2001 IEEE International Fuzzy Systems Conference*, 2001.

[19] Viswanadham, N. and Narahari, Y., *Performance Modeling of Automated Systems*, Upper Saddle River, NJ: Prentice Hall, Inc., 1992.

[20] Yaman, H., Karasan, O. E., and Pinar, M. C., "The Robust Spanning Tree Problem with Interval Data," *Operation Research Letters*, vol. 29, pp. 31-40, 2000.

APPENDIXES

APPENDIX A

AVERY SHORTEST PATH ALGORITHM
VISUAL BASIC CODE

```
Dim AllowedHigh As Integer
Dim AllowedLow As Integer
Dim ArcLower(1 To 25, 1 To 25) As Integer
Dim ArcsInShortPath As Integer
Dim ArcUpper(1 To 25, 1 To 25) As Integer
Dim Check(1 To 25, 1 To 25, 1 To 25, 1 To 25) As Boolean
Dim CombinedPath(1 To 100, 1 To 1000) As Integer
'CombinedPath(ComboNumber, PathNumber)
Dim CombinedSubPath(1 To 25, 1 To 25, 1 To 25, 1 To 100) As Integer
'CombinedSubPath(SubPathLength, NumberOfCombinedSubPaths(SubPathLength),
SubPosition, Number)
Dim ComboEndThreads(1 To 100, 1 To 200) As Integer
'ComboEndThreads(ComboNumber, i)
Dim ComboNumber As Integer
Dim ComboNumberNonDominated(1 To 100) As Integer
'ComboNumberNonDominated(ComboNumber) = 1
Dim ComboPathFound(1 To 1000) As Integer
'ComboPathFound(PathNumber)
Dim ComboSubPathFound(1 To 25, 1 To 100)
'ComboSubPathFound(SubPathLength, Number)
Dim ComboPathLength(1 To 25) As Single
'ComboPathLength (ComboNumber)
Dim ComboPathPoints(1 To 100) As Single
Dim ComboSubPathMatch(1 To 25, 1 To 100, 1 To 100) As Integer
'ComboSubPathMatch(SubPathLength,SubPathComboNumber, SubPathComboNumber)
Dim CumulativeDistanceChange(1 To 25) As Integer
Dim CurrentHigh As Integer
Dim CurrentLow As Integer
Dim CutPointCardinality(1 To 25) As Integer
Dim CutPoints(1 To 25, 1 To 100) As Integer
Dim EndNode As Integer
Dim EndPathNode(1 To 1000, 1 To 25) As Integer
'EndPathNode(EndPathNumber, PathPosition) = PathNode(PathNumber, PathPosition,
Node, Thread)
Dim EndPathThread(1 To 1000, 1 To 25) As Integer
'EndPathThread(EndPathNumber, PathPosition) = PathThread(PathNumber,
PathPosition, Node, Thread)
Dim EndThread(1 To 1000) As Integer
'EndThread(EndPathNumber) = ThreadDim FromNode As Integer
```

```
Dim FromNode As Integer
Dim FromNodej As Integer
Dim FromNodePathNumber As Integer
Dim FromNodePrev As Integer
Dim FromNodeThread As Integer
Dim FromNodeThreadPrev As Integer
Dim GoodPath(1 To 1000) As Boolean
Dim HighThreadComboNumberPosition(1 To 100, 1 To 25) As Integer
'HighThreadComboNumberPosition(ComboNumber,Position)
Dim i As Integer
Dim j As Integer
Dim Infinity As Integer
Dim k As Integer
Dim LargestMaximum As Integer
Dim LargestMaximumCombo As Integer
Dim LastPathPosition(1 To 1000, 1 To 25, 1 To 25) As Integer
'LastPathPosition(PathNumber,ToNode,ToNodeThread)
Dim LastPathPositionEnd(1 To 1000) As Integer
'LastPathPositionEnd(EndPathNumber)
Dim LastPathPositionEndCombo(1 To 100) As Integer
'LastPathPositionEndCombo(ComboNumber) =
LastPathPositionEnd(ComboNumber(PositionInCombo))
Dim LowArcValue(1 To 25, 1 To 25, 1 To 25, 1 To 25) As Single
Dim LowerArcCompleteCombo(1 To 25, 1 To 25) As Single
Dim LowerArcPositionInCombo(1 To 25, 1 To 200, 1 To 25) As Single
'LowerArcPositionInCombo(ComboNumber,PositionInCombo, Position)
Dim LowThreadComboNumberPosition(1 To 100, 1 To 25) As Integer
'LowThreadComboNumberPosition(ComboNumber,Position
Dim MaximumPathDistance(1 To 25) As Integer
'MaximumPathDistance(ComboNumber) = 0
Dim MaxIterativeChange As Integer
Dim MaxPaths As Integer
Dim MaxPathLength As Integer
Dim MaxThreads As Integer
Dim MinimizedRegretCombo As Integer
'MinimizedRegretCombo = ComboNumber
Dim MinimumPathDistance(1 To 25) As Integer
'MinimumPathDistance(ComboNumber) = 0
Dim Multiplier As Integer
Dim NecessaryChange(1 To 25, 1 To 25, 1 To 25, 1 To 25) As Single
'NecChange(FromNodeThread, FromNode, ToNodeThread, ToNode)
Dim Node As Integer
Dim NodeValueLower(1 To 25, 1 To 25) As Single
Dim NodeValueUpper(1 To 25, 1 To 25) As Single
Dim NumberMatches(1 To 1000) As Integer
'NumberMatches(PathNumber)
```

```
Dim NumberOfArcs As Integer
Dim NumberOfArcsProceeding(1 To 25) As Integer
Dim NumberOfCombinedPaths As Integer
'NumberOfCombinedSubPaths(SubPathLength) =
NumberOfCombinedSubPaths(SubPathLength) + 1
Dim NumberOfCombinedSubPaths(1 To 25) As Integer
Dim NumberOfPaths(1 To 25, 1 To 25) As Integer
'NumberOfPaths(ToNode,ToNodeThread)
Dim NumberOfPathsContainingArc(1 To 100) As Integer
'NumberOfPathsContainingArc (i)
Dim NumberOfSubPaths(1 To 25) As Integer
'NumberofSubPaths(SubPathLength)
Dim NumberSubMatches(1 To 25, 1 To 1000) As Integer
Dim NumberUnOrdered(1 To 25) As Integer
Dim OriginatingNode(1 To 100) As Integer
'OriginatingNode(NumberOfArcs) = FromNode
Dim PathArc(1 To 1000, 1 To 25) As Integer
'PathArc(PathNumber, Position - 1)
Dim PathLowArcValue(1 To 25, 1 To 25, 1 To 1000, 1 To 25) As Single
'PathLowArcValue(From Node,ToNode,CumulativePathNumber,Position)
Dim PathMatch(1 To 1000, 1 To 1000) As Integer
'PathMatch(PathNumber, j)
Dim PathNode(1 To 1000, 1 To 25, 1 To 25, 1 To 25) As Integer
Dim PathNumber As Integer
Dim PathPoints(1 To 1000) As Single
'PathPoints (PathNumber)
Dim PathsInCombo(1 To 1000) As Integer
'PathsInCombo(ComboNumber)
Dim PathThread(1 To 1000, 1 To 25, 1 To 25, 1 To 25) As Integer
Dim PathUpperArcValue(1 To 25, 1 To 25, 1 To 1000, 1 To 25) As Single
'PathUpperArcValue(From Node,ToNode,CumulativePathNumber,Position)
Dim Position As Integer
Dim PositionGood(1 To 1000) As Boolean
'PositionGood(PathNumber) = False
Dim PositionInCombo As Integer
Dim PositionLength(1 To 25, 1 To 25) As Single
'PositionLength(ComboNumber,Position)
Dim PositionNext As Integer
Dim RangeCalc As Integer
Dim Range As Integer
Dim RelativeWorthArc(1 To 100) As Single
Dim ShortestPathLength(1 To 25) As Single
'ShortestPathLength(ComboNumber)
Dim SmallestChange As Integer
'SmallestChange = CumulativeDistanceChange(ComboNumber)
Dim SmallestMinimum As Integer
```

```
Dim SmallestMinimumCombo As Integer
Dim StoppingCutPoint(1 To 25) As Integer
Dim StoppingPosition(1 To 25) As Integer
Dim SubPath(1 To 25, 1 To 1000, 1 To 25) As Integer
'SubPath(Length, Number, Position) As Integer
Dim SubPathComboNumber(1 To 25, 1 To 1000) As Integer
'SubPathComboNumber(Length, Number) As Integer
Dim SubPathLength As Integer
Dim SubPathMatch(1 To 25, 1 To 100, 1 To 100) As Integer
'SubPathMatch(SubPathLength,PathNumber,j)
Dim SubPosition As Integer
Dim SubsequentChange(1 To 25, 1 To 25, 1 To 25, 1 To 25) As Single
Dim t As Integer
Dim Temp As Integer
Dim Temp2 As Integer
Dim TerminatingNode(1 To 100) As Integer
'TerminatingNode(NumberOfArcs) = ToNode
Dim Thread As Integer
Dim ThreadInComboNumber(1 To 100, 1 To 100) As Integer
'ThreadInComboNumber(ComboNumber, Thread) = 1
Dim ToNode As Integer
Dim ToNodej As Integer
Dim ToNodePathNumber As Integer
Dim ToNodePrev As Integer
Dim ToNodeThread As Integer
Dim ToNodeThreadPrev As Integer
Dim TotalNumberOfPaths As Integer
Dim UnorderedCutPoints(1 To 75, 1 To 75) As Integer
Dim UpperArcCompleteCombo(1 To 25, 1 To 25) As Single
'UpperArcCompleteCombo(ComboNumber, Position)
Dim UpperArcPositionInCombo(1 To 25, 1 To 200, 1 To 25) As Single
Dim UpperArcValue(1 To 25, 1 To 25, 1 To 25, 1 To 25) As Single
Dim WCardinality(1 To 25) As Integer
Dim z As Integer

Sub AveryShortestPath()

ClearData
ReadData

For Node = 2 To EndNode
   IdentifyCutPoints
   OrderCutPoints
   GenerateNodeValue
Next Node
'loop = n
```

```
FindMultiplier

NumberOfArcs = 0
RangeCalc = 0
For FromNode = 1 To EndNode
  For ToNode = 1 To EndNode
    If ArcLower(FromNode, ToNode) < Infinity Then
      RangeCalc = RangeCalc + 1
      CheckPossibleArcs
    End If
  Next ToNode
Next FromNode

'loop = n ^ 2

FindPathThreads
IdentifyPathLowHigh
ChangeArcValuesOnPath
CheckPaths
NumberPaths
WritePaths

''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''OUTPUT METHODOLOGY''''''''''''''''''''''''


CheckIdenticalPaths
CombinePathThreads
ObtainComboThreadInfo
FindArcValuesOnCombinedPath

FindNonDominatedPaths
MinimizeRegret

GetArcs
EvaluateDominantArcs
WriteArcInfo

FindSubPaths
CheckIdenticalSubPaths
FindCombosInSubPaths

End Sub

Sub ClearData()
```

```
Worksheets("Network").Activate

Worksheets("NodeThreads").Activate
Worksheets("PossibleArcs").Activate
Worksheets("DominantArcs").Activate
Worksheets("ArcInfo").Activate
Worksheets("UnCorrectedPathThreads").Activate
Worksheets("PathThreads").Activate
Worksheets("NonDominatedPaths").Activate
Worksheets("CombinedPaths").Activate
Worksheets("SubPaths").Activate
Worksheets("NodeThreads").Range("A1:IV1250").Clear
Worksheets("PossibleArcs").Range("A1:IV1250").Clear
Worksheets("DominantArcs").Range("A1:IV1250").Clear
Worksheets("ArcInfo").Range("A1:IV1250").Clear
Worksheets("UnCorrectedPathThreads").Range("A1:IV1250").Clear
Worksheets("PathThreads").Range("A1:IV1250").Clear
Worksheets("CombinedPaths").Range("A1:IV1250").Clear
Worksheets("NonDominatedPaths").Range("A1:IV1250").Clear
Worksheets("SubPaths").Range("A1:IV1250").Clear

Infinity = 999

MaxNodes = 25
MaxThreads = 25
MaxPaths = 180
ToNode = 0
FromNode = 0
'FromNodeThread = 0
'ToNodeThread = 0
RangeCalc = 0
i = 0
z = 0
j = 0
k = 0
t = 0
EndNode = 0
Node = 0
Temp = 0
For i = 1 To MaxNodes
CutPointCardinality(i) = 0
WCardinality(i) = 1
NumberUnOrdered(i) = 0
StoppingCutPoint(i) = 0
StoppingPosition(i) = 0
```

```
NumberMatches(i) = 0
NumberOfPathsContainingArc(i) = 0
For j = 1 To MaxThreads
   ThreadInComboNumber(i, j) = 0
   NodeValueLower(i, j) = 0
   NodeValueUpper(i, j) = 0
   NumberOfPaths(i, j) = 0
   ArcLower(i, j) = 0
   ArcUpper(i, j) = 0
   UnorderedCutPoints(i, j) = 0
   CutPoints(i, j) = 0
   For k = 1 To MaxPaths
        LastPathPosition(k, i, j) = 0
   Next k
Next j
Next i

End Sub

Sub ReadData()

EndNode = Worksheets("Network").Cells(1, 2)

For i = 1 To EndNode
For j = 1 To EndNode
   ArcLower(i, j) = Worksheets("Network").Cells(i + 2, j + 1)
   ArcUpper(i, j) = Worksheets("Network").Cells(i + 29, j + 1)
   If ArcLower(i, j) < Infinity Then NumberOfArcsProceeding(j) =
NumberOfArcsProceeding(j) + 1
Next j
Next i

'loop = n ^ 2
End Sub

Sub IdentifyCutPoints()

StoppingCutPoint(Node) = Infinity
k = 1
For i = 1 To EndNode
   For t = 1 To WCardinality(i)
      UnorderedCutPoints(Node, k) = ArcLower(i, Node) + NodeValueLower(i, t)
      k = k + 1
   Next t
Next i
'max wcardinality (i) = 2^(i-2) i>3 sum of maz = 2^(n-1)
```

```
' max loop = n*2^(n-1)

For i = 1 To EndNode
      For t = 1 To WCardinality(i)
      UnorderedCutPoints(Node, k) = ArcUpper(i, Node) + NodeValueUpper(i, t)
      If t = WCardinality(i) And (UnorderedCutPoints(Node, k) <
StoppingCutPoint(Node)) And (UnorderedCutPoints(Node, k) < Infinity) Then
         StoppingCutPoint(Node) = UnorderedCutPoints(Node, k)
         ' This identifies the smallest of the NewUpperArcValues.
         ' Any cut points above this number will not be in the set of node threads.
         ' Node threads will consist of the set of all cut points smaller than or equal to the
stopping cut point.
      ' the stopping cut point must be the smallest of the new upper arc values(Omegas)
from among only the
      'Omegas that came from the end thread to ensure all possible shortest paths
      End If
      k = k + 1
   Next t
Next i
' max loop = n*2^(n-1)

NumberUnOrdered(Node) = k - 1

'For i = 1 To k - 1
'Worksheets("NodeThreads").Cells(i + 4, 10 + Node) = UnorderedCutPoints(Node,i)
'Next i

End Sub

Sub OrderCutPoints()

Worksheets("NodeThreads").Cells(1, 1) = "StoppingCutPoint"
Worksheets("NodeThreads").Cells(2, 1) = "CutPointCardinality"
Worksheets("NodeThreads").Cells(3, 1) = "StoppingPosition"
Worksheets("NodeThreads").Cells(4, 1) = "NumberOfThreads"
Worksheets("NodeThreads").Cells(5, 1) = "CUT POINTS"


Temp = Infinity
StoppingPosition(Node) = 1
For i = 1 To NumberUnOrdered(Node)
   If UnorderedCutPoints(Node, i) < Temp Then
   Temp = UnorderedCutPoints(Node, i)
   End If
Next i
CutPoints(Node, 1) = Temp
```

```
For z = 2 To NumberUnOrdered(Node)
   Temp = Infinity
   For i = 1 To NumberUnOrdered(Node)
      If (UnorderedCutPoints(Node, i) < Temp) And (UnorderedCutPoints(Node, i) >
CutPoints(Node, z - 1)) Then
      Temp = UnorderedCutPoints(Node, i)
      CutPointCardinality(Node) = z
      End If
   Next i
If Temp = Infinity Then
   CutPointCardinality(Node) = z - 1
   z = NumberUnOrdered(Node)
   'Ends LOOP
Else: CutPoints(Node, z) = Temp
   If CutPoints(Node, z) = StoppingCutPoint(Node) Then
     StoppingPosition(Node) = z
   End If
End If
Next z

For i = 1 To CutPointCardinality(Node)
   Worksheets("NodeThreads").Cells(i + 4, Node + 1) = CutPoints(Node, i)
Next i

If StoppingPosition(Node) = 1 Then
   WCardinality(Node) = 1
Else
   WCardinality(Node) = StoppingPosition(Node) - 1
End If


Worksheets("NodeThreads").Cells(1, 1 + Node) = StoppingCutPoint(Node)
Worksheets("NodeThreads").Cells(2, 1 + Node) = CutPointCardinality(Node)
Worksheets("NodeThreads").Cells(3, Node + 1) = StoppingPosition(Node)
Worksheets("NodeThreads").Cells(4, Node + 1) = WCardinality(Node)

End Sub

Sub GenerateNodeValue()

k = 0
For Thread = 1 To WCardinality(Node)
   k = k + 2
   If (StoppingPosition(Node) = 1) Then
      NodeValueLower(Node, Thread) = CutPoints(Node, 1)
```

```
      NodeValueUpper(Node, Thread) = CutPoints(Node, 1)
   Else
      NodeValueLower(Node, Thread) = CutPoints(Node, Thread)
      NodeValueUpper(Node, Thread) = CutPoints(Node, Thread + 1)
   End If
   Worksheets("NodeThreads").Cells(k - 1, 5 + EndNode + Node) =
NodeValueLower(Node, Thread)
   Worksheets("NodeThreads").Cells(k, 5 + EndNode + Node) = NodeValueUpper(Node,
Thread)
   Worksheets("NodeThreads").Cells(2 * (Thread - 1) + 1, 5 + EndNode) =
"NodeValueLower"
   Worksheets("NodeThreads").Cells(2 * (Thread - 1) + 2, 5 + EndNode) =
"NodeValueUpper"
Next Thread

End Sub

Sub FindMultiplier()

MaxThreads = 0
For Node = 1 To EndNode
   If WCardinality(Node) > MaxThreads Then
      MaxThreads = WCardinality(Node)
   End If
Next Node

Multiplier = 10
For i = 1 To 5
If MaxThreads > 10 ^ i - 1 Then
Multiplier = 10 ^ (i + 1)
End If
Next i

End Sub

Sub CheckPossibleArcs()

For ToNodeThread = 1 To WCardinality(ToNode)
For FromNodeThread = 1 To WCardinality(FromNode)
   LowArcValue(FromNode, FromNodeThread, ToNode, ToNodeThread) =
NodeValueLower(ToNode, ToNodeThread) - NodeValueLower(FromNode,
FromNodeThread)
   UpperArcValue(FromNode, FromNodeThread, ToNode, ToNodeThread) =
NodeValueUpper(ToNode, ToNodeThread) - NodeValueUpper(FromNode,
FromNodeThread)
   RangeCalc = RangeCalc + 1
```

```
   Worksheets("PossibleArcs").Cells(1, 1) = "TO"
   Worksheets("PossibleArcs").Cells(1, 2) = "FROM"

   Worksheets("PossibleArcs").Cells(1, 4 + FromNode) = FromNode
   Worksheets("PossibleArcs").Cells(1, 4 + 1 * EndNode + FromNode) = FromNode
   Worksheets("PossibleArcs").Cells(1, 4 + 2 * EndNode + FromNode) = FromNode
   Worksheets("PossibleArcs").Cells(1, 4 + 3 * EndNode + FromNode) = FromNode

   Worksheets("PossibleArcs").Cells(RangeCalc, 4 + FromNode) =
LowArcValue(FromNode, FromNodeThread, ToNode, ToNodeThread)
   Worksheets("PossibleArcs").Cells(RangeCalc, 4 + 1 * EndNode + FromNode) =
UpperArcValue(FromNode, FromNodeThread, ToNode, ToNodeThread)
   Worksheets("PossibleArcs").Cells(RangeCalc, 4 + 2 * EndNode + FromNode) =
ArcLower(FromNode, ToNode)
   Worksheets("PossibleArcs").Cells(RangeCalc, 4 + 3 * EndNode + FromNode) =
ArcUpper(FromNode, ToNode)
   Worksheets("PossibleArcs").Cells(RangeCalc, 1) = FromNode * Multiplier +
FromNodeThread
   Worksheets("PossibleArcs").Cells(RangeCalc, 2) = ToNode * Multiplier +
ToNodeThread


   'Worksheets("PossibleArcs").Cells(1, 1) = "TO"
   'Worksheets("PossibleArcs").Cells(2, 1) = "FROM"
   'Worksheets("PossibleArcs").Cells(4 + FromNode, 1) = FromNode
   'Worksheets("PossibleArcs").Cells(4 + 1 * EndNode + FromNode, 1) = FromNode
   'Worksheets("PossibleArcs").Cells(4 + 2 * EndNode + FromNode, 1) = FromNode
   'Worksheets("PossibleArcs").Cells(4 + 3 * EndNode + FromNode, 1) = FromNode

AllowedLow = ArcLower(FromNode, ToNode)
AllowedHigh = ArcUpper(FromNode, ToNode)
CurrentLow = LowArcValue(FromNode, FromNodeThread, ToNode, ToNodeThread)
CurrentHigh = UpperArcValue(FromNode, FromNodeThread, ToNode, ToNodeThread)
If CurrentLow >= AllowedLow And CurrentHigh <= AllowedHigh Then
   Check(FromNode, FromNodeThread, ToNode, ToNodeThread) = True
   Worksheets("PossibleArcs").Cells(RangeCalc, 3) = 0
If CurrentLow > CurrentHigh Then Worksheets("PossibleArcs").Cells(4, 1) = "C"
Else
Check(FromNode, FromNodeThread, ToNode, ToNodeThread) = False
Worksheets("PossibleArcs").Cells(RangeCalc, 3) = "x"
End If

Next FromNodeThread
Next ToNodeThread
```

```
End Sub

Sub FindPathThreads()

RangeCalc = 0

For ToNode = 2 To EndNode
For ToNodeThread = 1 To WCardinality(ToNode)
   If Check(1, 1, ToNode, ToNodeThread) = True Then
      PathNode(1, 1, ToNode, ToNodeThread) = 1
      PathNode(1, 2, ToNode, ToNodeThread) = ToNode
      PathThread(1, 1, ToNode, ToNodeThread) = 1
      PathThread(1, 2, ToNode, ToNodeThread) = ToNodeThread
      NumberOfPaths(ToNode, ToNodeThread) = NumberOfPaths(ToNode,
ToNodeThread) + 1
      LastPathPosition(1, ToNode, ToNodeThread) = 2
   End If
For FromNode = 2 To ToNode - 1
   For FromNodeThread = 1 To WCardinality(FromNode)
   If Check(FromNode, FromNodeThread, ToNode, ToNodeThread) = True Then
      For i = 1 To NumberOfPaths(FromNode, FromNodeThread)
       For j = 1 To LastPathPosition(i, FromNode, FromNodeThread)
         PathNode(NumberOfPaths(ToNode, ToNodeThread) + i, j, ToNode,
ToNodeThread) = PathNode(i, j, FromNode, FromNodeThread)
         PathThread(NumberOfPaths(ToNode, ToNodeThread) + i, j, ToNode,
ToNodeThread) = PathThread(i, j, FromNode, FromNodeThread)
      Next j
      PathNode(NumberOfPaths(ToNode, ToNodeThread) + i, LastPathPosition(i,
FromNode, FromNodeThread) + 1, ToNode, ToNodeThread) = ToNode
      PathThread(NumberOfPaths(ToNode, ToNodeThread) + i, LastPathPosition(i,
FromNode, FromNodeThread) + 1, ToNode, ToNodeThread) = ToNodeThread
      LastPathPosition(NumberOfPaths(ToNode, ToNodeThread) + i, ToNode,
ToNodeThread) = LastPathPosition(i, FromNode, FromNodeThread) + 1
   Next i
   NumberOfPaths(ToNode, ToNodeThread) = NumberOfPaths(ToNode,
ToNodeThread) + NumberOfPaths(FromNode, FromNodeThread)
   End If
   Next FromNodeThread
   Next FromNode
Next ToNodeThread
Next ToNode

RangeCalc = 0
EndPathNumber = 0
TotalNumberOfPaths = 0
MaxPathLength = 0
```

```
Node = EndNode
For Thread = 1 To WCardinality(Node)
For PathNumber = 1 To NumberOfPaths(Node, Thread)
   RangeCalc = RangeCalc + 1
      EndPathNumber = EndPathNumber + 1
   TotalNumberOfPaths = TotalNumberOfPaths + 1
   For PathPosition = 1 To LastPathPosition(PathNumber, Node, Thread)
      If LastPathPosition(PathNumber, Node, Thread) > MaxPathLength Then
MaxPathLength = LastPathPosition(PathNumber, Node, Thread)
      EndPathNode(EndPathNumber, PathPosition) = PathNode(PathNumber,
PathPosition, Node, Thread)
      EndPathThread(EndPathNumber, PathPosition) = PathThread(PathNumber,
PathPosition, Node, Thread)
      EndThread(EndPathNumber) = Thread
      Worksheets("UnCorrectedPathThreads").Cells(1 + RangeCalc, 1 + PathPosition) =
(PathNode(PathNumber, PathPosition, Node, Thread)) * Multiplier +
PathThread(PathNumber, PathPosition, Node, Thread)
   Next PathPosition
   LastPathPositionEnd(EndPathNumber) = LastPathPosition(PathNumber, Node,
Thread)
   Worksheets("UnCorrectedPathThreads").Cells(1 + EndPathNumber, 11) =
LastPathPositionEnd(EndPathNumber)
   Worksheets("UnCorrectedPathThreads").Cells(1 + EndPathNumber, 13) =
EndPathNumber

  Next PathNumber
Next Thread

End Sub

Sub IdentifyPathLowHigh()

For PathNumber = 1 To TotalNumberOfPaths
For Position = LastPathPositionEnd(PathNumber) To 2 Step -1

ToNode = EndPathNode(PathNumber, Position)
FromNode = EndPathNode(PathNumber, Position - 1)
ToNodeThread = EndPathThread(PathNumber, Position)
FromNodeThread = EndPathThread(PathNumber, Position - 1)

PathLowArcValue(FromNode, ToNode, PathNumber, Position) =
NodeValueLower(ToNode, ToNodeThread) - NodeValueLower(FromNode,
FromNodeThread)
PathUpperArcValue(FromNode, ToNode, PathNumber, Position) =
NodeValueUpper(ToNode, ToNodeThread) - NodeValueUpper(FromNode,
FromNodeThread)
```

```
Next Position
Next PathNumber

RangeCalc = 1
For PathNumber = 1 To TotalNumberOfPaths

Worksheets("UnCorrectedPathThreads").Cells(2 + TotalNumberOfPaths + RangeCalc,
10) = _
   NodeValueLower(EndPathNode(PathNumber, LastPathPositionEnd(PathNumber)),
EndPathThread(PathNumber, LastPathPositionEnd(PathNumber)))

Worksheets("UnCorrectedPathThreads").Cells(2 + TotalNumberOfPaths + RangeCalc,
11) = _
   NodeValueUpper(EndPathNode(PathNumber, LastPathPositionEnd(PathNumber)),
EndPathThread(PathNumber, LastPathPositionEnd(PathNumber)))

For Position = LastPathPositionEnd(PathNumber) To 2 Step -1

ToNode = EndPathNode(PathNumber, Position)
FromNode = EndPathNode(PathNumber, Position - 1)
ToNodeThread = EndPathThread(PathNumber, Position)
FromNodeThread = EndPathThread(PathNumber, Position - 1)


Worksheets("UnCorrectedPathThreads").Cells(2 + TotalNumberOfPaths + RangeCalc, 2)
= ArcLower(FromNode, ToNode)
Worksheets("UnCorrectedPathThreads").Cells(2 + TotalNumberOfPaths + RangeCalc, 3)
= PathLowArcValue(FromNode, ToNode, PathNumber, Position)
Worksheets("UnCorrectedPathThreads").Cells(2 + TotalNumberOfPaths + RangeCalc, 4)
= PathUpperArcValue(FromNode, ToNode, PathNumber, Position)
Worksheets("UnCorrectedPathThreads").Cells(2 + TotalNumberOfPaths + RangeCalc, 5)
= ArcUpper(FromNode, ToNode)
Worksheets("UnCorrectedPathThreads").Cells(2 + TotalNumberOfPaths + RangeCalc, 7)
= FromNode * Multiplier + FromNodeThread
Worksheets("UnCorrectedPathThreads").Cells(2 + TotalNumberOfPaths + RangeCalc, 8)
= ToNode * Multiplier + ToNodeThread
RangeCalc = RangeCalc + 1

Next Position
RangeCalc = RangeCalc + 1

Next PathNumber
RangeCalc = RangeCalc + 1

End Sub
```

```
Sub ChangeArcValuesOnPath()

RangeCalc = 0
For PathNumber = 1 To TotalNumberOfPaths
For Position = LastPathPositionEnd(PathNumber) To 2 Step -1

ToNode = EndPathNode(PathNumber, Position)
FromNode = EndPathNode(PathNumber, Position - 1)
ToNodeThread = EndPathThread(PathNumber, Position)
FromNodeThread = EndPathThread(PathNumber, Position - 1)

CurrentLow = PathLowArcValue(FromNode, ToNode, PathNumber, Position)
CurrentHigh = PathUpperArcValue(FromNode, ToNode, PathNumber, Position)

AllowedLow = ArcLower(FromNode, ToNode)
AllowedHigh = ArcUpper(FromNode, ToNode)

If (CurrentLow > CurrentHigh) Then
 If ((CurrentLow > AllowedHigh) And (CurrentHigh >= AllowedLow)) Then
   'We are decreasing the PathLowArcValue
   NecessaryChange(FromNode, FromNodeThread, ToNode, ToNodeThread) = _
   PathLowArcValue(FromNode, ToNode, PathNumber, Position) -
PathUpperArcValue(FromNode, ToNode, PathNumber, Position)
   For PositionNext = Position - 1 To 2 Step -1
      ToNodePrev = EndPathNode(PathNumber, PositionNext)
      FromNodePrev = EndPathNode(PathNumber, PositionNext - 1)
      ToNodeThreadPrev = EndPathThread(PathNumber, PositionNext)
      FromNodeThreadPrev = EndPathThread(PathNumber, PositionNext - 1)
      'We are increasing subsequent lower paths as much as possible
      MaxIterativeChange = _
         PathUpperArcValue(FromNodePrev, ToNodePrev, PathNumber, PositionNext) -
PathLowArcValue(FromNodePrev, ToNodePrev, PathNumber, PositionNext)
      If MaxIterativeChange < 0 Then
         MaxIterativeChange = 0
      End If
  If MaxIterativeChange >= NecessaryChange(FromNode, FromNodeThread, ToNode,
ToNodeThread) Then
      'increase by average of necessary and max
      PathLowArcValue(FromNodePrev, ToNodePrev, PathNumber, PositionNext) = _
         PathLowArcValue(FromNodePrev, ToNodePrev, PathNumber, PositionNext) _
         + NecessaryChange(FromNode, FromNodeThread, ToNode, ToNodeThread)
      PathLowArcValue(FromNode, ToNode, PathNumber, Position) = _
         PathLowArcValue(FromNode, ToNode, PathNumber, Position) _
         - NecessaryChange(FromNode, FromNodeThread, ToNode, ToNodeThread)
      NecessaryChange(FromNode, FromNodeThread, ToNode, ToNodeThread) = 0
```

```
  Else
     'decrease upper by Max
     PathLowArcValue(FromNodePrev, ToNodePrev, PathNumber, PositionNext) = _
        PathLowArcValue(FromNodePrev, ToNodePrev, PathNumber, PositionNext) _
       + MaxIterativeChange
     PathLowArcValue(FromNode, ToNode, PathNumber, Position) = _
        PathLowArcValue(FromNode, ToNode, PathNumber, Position) _
        - MaxIterativeChange
     NecessaryChange(FromNode, FromNodeThread, ToNode, ToNodeThread) = _
        NecessaryChange(FromNode, FromNodeThread, ToNode, ToNodeThread) _
        - MaxIterativeChange
  End If
  If NecessaryChange(FromNode, FromNodeThread, ToNode, ToNodeThread) = 0 Then
    PositionNext = 2
    'endloop of IncreasingSubsequentLower
  End If
  Next PositionNext

Else

'If CurrentHigh < AllowedLow Then
   'We are increasing the PathUpperArcValue
   NecessaryChange(FromNode, FromNodeThread, ToNode, ToNodeThread) = _
   PathLowArcValue(FromNode, ToNode, PathNumber, Position) -
PathUpperArcValue(FromNode, ToNode, PathNumber, Position)
   For PositionNext = Position - 1 To 2 Step -1
      ToNodePrev = EndPathNode(PathNumber, PositionNext)
      FromNodePrev = EndPathNode(PathNumber, PositionNext - 1)
      ToNodeThreadPrev = EndPathThread(PathNumber, PositionNext)
      FromNodeThreadPrev = EndPathThread(PathNumber, PositionNext - 1)
      'We are decreasing subsequent upper paths as much as possible
      MaxIterativeChange = _
         PathUpperArcValue(FromNodePrev, ToNodePrev, PathNumber, PositionNext) -
PathLowArcValue(FromNodePrev, ToNodePrev, PathNumber, PositionNext)
      If MaxIterativeChange < 0 Then
         MaxIterativeChange = 0
      End If
  If MaxIterativeChange >= NecessaryChange(FromNode, FromNodeThread, ToNode,
ToNodeThread) Then
      'decrease by average of necessary and max
      PathUpperArcValue(FromNodePrev, ToNodePrev, PathNumber, PositionNext) = _
         PathUpperArcValue(FromNodePrev, ToNodePrev, PathNumber, PositionNext) _
         - (NecessaryChange(FromNode, FromNodeThread, ToNode, ToNodeThread))
      PathUpperArcValue(FromNode, ToNode, PathNumber, Position) = _
         PathUpperArcValue(FromNode, ToNode, PathNumber, Position) _
         + (NecessaryChange(FromNode, FromNodeThread, ToNode, ToNodeThread))
```

```
    NecessaryChange(FromNode, FromNodeThread, ToNode, ToNodeThread) = 0
  Else
    'decrease upper by Max
    PathUpperArcValue(FromNodePrev, ToNodePrev, PathNumber, PositionNext) = _
      PathUpperArcValue(FromNodePrev, ToNodePrev, PathNumber, PositionNext) _
      - MaxIterativeChange
    PathUpperArcValue(FromNode, ToNode, PathNumber, Position) = _
      PathUpperArcValue(FromNode, ToNode, PathNumber, Position) _
      + MaxIterativeChange
    NecessaryChange(FromNode, FromNodeThread, ToNode, ToNodeThread) = _
      NecessaryChange(FromNode, FromNodeThread, ToNode, ToNodeThread) _
      - MaxIterativeChange
  End If
  If NecessaryChange(FromNode, FromNodeThread, ToNode, ToNodeThread) = 0 Then
      PositionNext = 2
      'endloop of IncreasingSubsequentLower
  End If
  Next PositionNext
  End If
End If
RangeCalc = RangeCalc + 1

Worksheets("PathThreads").Cells(152 + RangeCalc, 2) = PathNumber
Worksheets("PathThreads").Cells(152 + RangeCalc, 3) = ArcLower(FromNode,
ToNode)
Worksheets("PathThreads").Cells(152 + RangeCalc, 4) = PathLowArcValue(FromNode,
ToNode, PathNumber, Position)
Worksheets("PathThreads").Cells(152 + RangeCalc, 5) =
PathUpperArcValue(FromNode, ToNode, PathNumber, Position)
Worksheets("PathThreads").Cells(152 + RangeCalc, 6) = ArcUpper(FromNode, ToNode)
Worksheets("PathThreads").Cells(152 + RangeCalc, 8) = FromNode * Multiplier +
FromNodeThread
Worksheets("PathThreads").Cells(152 + RangeCalc, 9) = ToNode * Multiplier +
ToNodeThread


Next Position
RangeCalc = RangeCalc + 1

Next PathNumber

End Sub

Sub CheckPaths()

For PathNumber = 1 To TotalNumberOfPaths
```

```
SumLower = 0
SumUpper = 0
PositionGood(PathNumber) = True
For Position = LastPathPositionEnd(PathNumber) To 2 Step -1
   ToNode = EndPathNode(PathNumber, Position)
   FromNode = EndPathNode(PathNumber, Position - 1)
   ToNodeThread = EndPathThread(PathNumber, Position)
   FromNodeThread = EndPathThread(PathNumber, Position - 1)
   SumLower = SumLower + PathLowArcValue(FromNode, ToNode, PathNumber,
Position)
   SumUpper = SumUpper + PathUpperArcValue(FromNode, ToNode, PathNumber,
Position)
   If PathLowArcValue(FromNode, ToNode, PathNumber, Position) >
PathUpperArcValue(FromNode, ToNode, PathNumber, Position) Then
PositionGood(PathNumber) = False
Next Position

If SumLower = NodeValueLower(EndNode, EndThread(PathNumber)) _
   And SumUpper = NodeValueUpper(EndNode, EndThread(PathNumber)) _
   And PositionGood(PathNumber) = True Then
GoodPath(PathNumber) = True
Else: GoodPath(PathNumber) = False
End If

Next PathNumber
End Sub

Sub NumberPaths()

RangeCalc = 0
EndPathNumber = 0
OldTotalNumberOfPaths = TotalNumberOfPaths
TotalNumberOfPaths = 0
'MaxPathLength = 0
Node = EndNode

For PathNumber = 1 To OldTotalNumberOfPaths
If GoodPath(PathNumber) = True Then
   RangeCalc = RangeCalc + 1
   EndPathNumber = EndPathNumber + 1
   TotalNumberOfPaths = TotalNumberOfPaths + 1
   LastPathPositionEnd(EndPathNumber) = LastPathPositionEnd(PathNumber)
   Worksheets("PathThreads").Cells(1 + EndPathNumber, 11) = EndPathNumber
   Worksheets("PathThreads").Cells(1 + EndPathNumber, 13) =
LastPathPositionEnd(EndPathNumber)
   Worksheets("PathThreads").Cells(1 + EndPathNumber, 9) = PathNumber
```

```
  For Position = LastPathPositionEnd(PathNumber) To 2 Step -1
     ToNode = EndPathNode(PathNumber, Position)
     FromNode = EndPathNode(PathNumber, Position - 1)
     ToNodeThread = EndPathThread(PathNumber, Position)
     FromNodeThread = EndPathThread(PathNumber, Position - 1)
     EndPathNode(EndPathNumber, Position) = EndPathNode(PathNumber, Position)
     EndPathThread(EndPathNumber, Position) = EndPathThread(PathNumber,
Position)
     PathLowArcValue(FromNode, ToNode, EndPathNumber, Position) =
PathLowArcValue(FromNode, ToNode, PathNumber, Position)
     PathUpperArcValue(FromNode, ToNode, EndPathNumber, Position) =
PathUpperArcValue(FromNode, ToNode, PathNumber, Position)
     Worksheets("PathThreads").Cells(1 + RangeCalc, 1 + Position) =
(EndPathNode(EndPathNumber, Position)) * Multiplier +
EndPathThread(EndPathNumber, Position)
  Next Position
End If
Next PathNumber

End Sub

Sub WritePaths()

RangeCalc = 0
For PathNumber = 1 To TotalNumberOfPaths
For Position = LastPathPositionEnd(PathNumber) To 2 Step -1
RangeCalc = RangeCalc + 1
  ToNode = EndPathNode(PathNumber, Position)
  FromNode = EndPathNode(PathNumber, Position - 1)
  ToNodeThread = EndPathThread(PathNumber, Position)
  FromNodeThread = EndPathThread(PathNumber, Position - 1)
Worksheets("PathThreads").Cells(152 + RangeCalc, 22) = PathNumber
Worksheets("PathThreads").Cells(152 + RangeCalc, 23) = ArcLower(FromNode,
ToNode)
Worksheets("PathThreads").Cells(152 + RangeCalc, 24) =
PathLowArcValue(FromNode, ToNode, PathNumber, Position)
Worksheets("PathThreads").Cells(152 + RangeCalc, 25) =
PathUpperArcValue(FromNode, ToNode, PathNumber, Position)
Worksheets("PathThreads").Cells(152 + RangeCalc, 26) = ArcUpper(FromNode,
ToNode)
Worksheets("PathThreads").Cells(152 + RangeCalc, 28) = FromNode * Multiplier +
FromNodeThread
Worksheets("PathThreads").Cells(152 + RangeCalc, 29) = ToNode * Multiplier +
ToNodeThread

Next Position
```

```
RangeCalc = RangeCalc + 1
Next PathNumber
End Sub

Sub CheckIdenticalPaths()

For PathNumber = 1 To TotalNumberOfPaths
NumberMatches(PathNumber) = 1
For j = 1 To TotalNumberOfPaths
   PathMatch(PathNumber, j) = 0
   If LastPathPositionEnd(PathNumber) = LastPathPositionEnd(j) Then
      PathMatch(PathNumber, j) = 1
      For Position = 2 To (LastPathPositionEnd(PathNumber) - 1)
         ToNodePathNumber = EndPathNode(PathNumber, Position)
         FromNodePathNumber = EndPathNode(PathNumber, Position - 1)
         ToNodej = EndPathNode(j, Position)
         FromNodej = EndPathNode(j, Position - 1)
         If ToNodePathNumber = ToNodej And FromNodePathNumber = FromNodej
And (PathMatch(PathNumber, j) = 1) Then

         'If (PathArc(PathNumber, Position) = PathArc(j, Position)) And
(PathMatch(PathNumber, j) = 1) Then
            PathMatch(PathNumber, j) = 1
         Else: PathMatch(PathNumber, j) = 0
         End If
      Next Position
   End If
   'Worksheets("DominantArcs").Cells(2 + PathNumber, 14 + j) =
PathMatch(PathNumber, j)
If (PathMatch(PathNumber, j) = 1) And (PathNumber <> j) Then
NumberMatches(PathNumber) = NumberMatches(PathNumber) + 1
Next j
Worksheets("DominantArcs").Cells(2 + PathNumber, 13) =
NumberMatches(PathNumber)
Next PathNumber

End Sub

Sub CombinePathThreads()

'look through paths combinedpaths(combonumber, pathnumber)

ComboNumber = 0
For PathNumber = 1 To TotalNumberOfPaths
   ComboPathFound(PathNumber) = 0
Next PathNumber
```

```
For PathNumber = 1 To TotalNumberOfPaths
   If ComboPathFound(PathNumber) = 0 Then
      ComboNumber = ComboNumber + 1
      CombinedPath(ComboNumber, 1) = PathNumber
      PathInPositionInCombo = 1
      For j = (PathNumber) To TotalNumberOfPaths
         If (PathMatch(PathNumber, j) = 1) And (PathNumber <> j) Then
            ComboPathFound(PathNumber) = 1
            ComboPathFound(j) = 1
            PathInPositionInCombo = PathInPositionInCombo + 1
            CombinedPath(ComboNumber, PathInPositionInCombo) = j
         End If
      PathsInCombo(ComboNumber) = PathInPositionInCombo
      Next j
   End If
Next PathNumber

NumberOfCombinedPaths = ComboNumber

End Sub

Sub ObtainComboThreadInfo()

'Initialize lowthread/highthread min/max path distance
For ComboNumber = 1 To NumberOfCombinedPaths
   MinimumPathDistance(ComboNumber) = 0
   MaximumPathDistance(ComboNumber) = 0
   For PositionInCombo = 1 To PathsInCombo(ComboNumber)
      For Position = LastPathPositionEnd(CombinedPath(ComboNumber, 1)) To 2 Step -
1
      LowThreadComboNumberPosition(ComboNumber, Position) = Infinity
      HighThreadComboNumberPosition(ComboNumber, Position) = 0
      Next Position
   Next PositionInCombo
Next ComboNumber

For ComboNumber = 1 To NumberOfCombinedPaths
   LowThreadComboNumberPosition(ComboNumber, 1) = 1
   HighThreadComboNumberPosition(ComboNumber, 1) = 1
   For PositionInCombo = 1 To PathsInCombo(ComboNumber)
   PathNumber = CombinedPath(ComboNumber, PositionInCombo)
   'Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1) +
1, 3 * (PositionInCombo - 1) + 25) = CombinedPath(ComboNumber, PositionInCombo)
   For Position = LastPathPositionEnd(PathNumber) To 2 Step -1
      ToNode = EndPathNode(PathNumber, Position)
```

```
      FromNode = EndPathNode(PathNumber, Position - 1)
      ToNodeThread = EndPathThread(PathNumber, Position)
      FromNodeThread = EndPathThread(PathNumber, Position)
      LowerArcPositionInCombo(ComboNumber, PositionInCombo, Position) =
PathLowArcValue(FromNode, ToNode, PathNumber, Position)
      UpperArcPositionInCombo(ComboNumber, PositionInCombo, Position) =
PathUpperArcValue(FromNode, ToNode, PathNumber, Position)
      'Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1)
+ Position, 3 * (PositionInCombo - 1) + 25) =
LowerArcPositionInCombo(ComboNumber, PositionInCombo, Position)
      'Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1)
+ Position, 3 * (PositionInCombo - 1) + 26) =
UpperArcPositionInCombo(ComboNumber, PositionInCombo, Position)
      'Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1)
+ Position, 6) = ArcLower(FromNode, ToNode)
      'Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1)
+ Position, 7) = ArcUpper(FromNode, ToNode)
      'Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1)
+ Position, 3) = FromNode
      'Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1)
+ Position, 4) = ToNode
      If ToNodeThread < LowThreadComboNumberPosition(ComboNumber, Position)
Then LowThreadComboNumberPosition(ComboNumber, Position) = ToNodeThread
      If ToNodeThread > HighThreadComboNumberPosition(ComboNumber, Position)
Then HighThreadComboNumberPosition(ComboNumber, Position) = ToNodeThread
   Next Position
   Next PositionInCombo
   Next ComboNumber

For ComboNumber = 1 To NumberOfCombinedPaths
   For Position = LastPathPositionEnd(CombinedPath(ComboNumber, 1)) To 2 Step -1
   ToNode = EndPathNode(CombinedPath(ComboNumber, 1), Position)
   FromNode = EndPathNode(CombinedPath(ComboNumber, 1), Position - 1)
   Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1) +
Position, 9) = _
                  LowThreadComboNumberPosition(ComboNumber, Position)
   Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1) +
Position, 10) = _
                  HighThreadComboNumberPosition(ComboNumber, Position)
   Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1) +
Position, 12) = _
                  NodeValueLower(ToNode,
LowThreadComboNumberPosition(ComboNumber, Position))
   Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1) +
Position, 13) = _
```

```
                NodeValueUpper(ToNode,
HighThreadComboNumberPosition(ComboNumber, Position))
  LowerArcCompleteCombo(ComboNumber, Position) = _
    NodeValueLower(ToNode, LowThreadComboNumberPosition(ComboNumber,
Position)) _
    - NodeValueLower(FromNode, LowThreadComboNumberPosition(ComboNumber,
Position - 1))
  UpperArcCompleteCombo(ComboNumber, Position) = _
    NodeValueUpper(ToNode, HighThreadComboNumberPosition(ComboNumber,
Position)) _
    - NodeValueUpper(FromNode, HighThreadComboNumberPosition(ComboNumber,
Position - 1))
  Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1) +
Position, 15) = _
                LowerArcCompleteCombo(ComboNumber, Position)
  Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1) +
Position, 16) = _
                UpperArcCompleteCombo(ComboNumber, Position)

  Next Position

Next ComboNumber

End Sub

Sub FindArcValuesOnCombinedPath()

For ComboNumber = 1 To NumberOfCombinedPaths
PathNumber = CombinedPath(ComboNumber, 1)
ComboPathLength(ComboNumber) = 0

If NodeValueUpper(EndNode, HighThreadComboNumberPosition(ComboNumber,
LastPathPositionEnd(PathNumber))) <> _
 NodeValueLower(EndNode, LowThreadComboNumberPosition(ComboNumber,
LastPathPositionEnd(PathNumber))) Then

ShortestPathLength(ComboNumber) = _
 NodeValueUpper(EndNode, HighThreadComboNumberPosition(ComboNumber,
LastPathPositionEnd(PathNumber))) _
- NodeValueLower(EndNode, LowThreadComboNumberPosition(ComboNumber,
LastPathPositionEnd(PathNumber)))

For Position = LastPathPositionEnd(CombinedPath(ComboNumber, 1)) To 2 Step -1

ToNode = EndPathNode(PathNumber, Position)
FromNode = EndPathNode(PathNumber, Position - 1)
```

```
ComboPathLength(ComboNumber) = ComboPathLength(ComboNumber) +
ArcUpper(FromNode, ToNode) - ArcLower(FromNode, ToNode)
Next Position

PositionLength(ComboNumber, 2) = ShortestPathLength(ComboNumber)

For Position = LastPathPositionEnd(CombinedPath(ComboNumber, 1)) To 3 Step -1
ToNode = EndPathNode(PathNumber, Position)
FromNode = EndPathNode(PathNumber, Position - 1)

PositionLength(ComboNumber, Position) = ShortestPathLength(ComboNumber) *
(ArcUpper(FromNode, ToNode) - ArcLower(FromNode, ToNode)) /
ComboPathLength(ComboNumber)
PositionLength(ComboNumber, 2) = PositionLength(ComboNumber, 2) -
PositionLength(ComboNumber, Position)

Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1) +
Position, 18) = _
    PositionLength(ComboNumber, Position)

UpperArcCompleteCombo(ComboNumber, Position) = _
  LowerArcCompleteCombo(ComboNumber, Position) +
PositionLength(ComboNumber, Position)

Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1) +
Position, 21) = _
    UpperArcCompleteCombo(ComboNumber, Position)
Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1) +
Position, 20) = _
    LowerArcCompleteCombo(ComboNumber, Position)

Next Position

UpperArcCompleteCombo(ComboNumber, 2) = _
  LowerArcCompleteCombo(ComboNumber, 2) + PositionLength(ComboNumber, 2)

End If

Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1) + 2,
18) = _
    PositionLength(ComboNumber, 2)
Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1) + 2,
21) = _
    UpperArcCompleteCombo(ComboNumber, 2)
Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1) + 2,
20) = _
```

```
    LowerArcCompleteCombo(ComboNumber, 2)

Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1) +
Position + 1, 17) = _
    ComboPathLength(ComboNumber)
Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1) +
Position + 2, 17) = _
    ShortestPathLength(ComboNumber)

Next ComboNumber
End Sub


Sub FindNonDominatedPaths()

' we are listing all end threads for each ComboNumber
Worksheets("NonDominatedPaths").Cells(1 * (NumberOfCombinedPaths + 2) + 4, 1) =
"NonDominatedCombos"
For ComboNumber = 1 To NumberOfCombinedPaths
  For PositionInCombo = 1 To PathsInCombo(ComboNumber)
    PathNumber = CombinedPath(ComboNumber, PositionInCombo)
    ThreadInComboNumber(ComboNumber, EndPathThread(PathNumber,
LastPathPositionEnd(PathNumber))) = 1
    ComboEndThreads(ComboNumber, PositionInCombo) =
EndPathThread(PathNumber, LastPathPositionEnd(PathNumber))
    Worksheets("NonDominatedPaths").Cells(2 * (NumberOfCombinedPaths + 2) +
ComboNumber + 4, 2 + PositionInCombo) = ComboEndThreads(ComboNumber,
PositionInCombo)
  Next PositionInCombo
Next ComboNumber

For ComboNumber = 1 To NumberOfCombinedPaths
ComboNumberNonDominated(ComboNumber) = 1
For Thread = 1 To WCardinality(EndNode)
  Worksheets("NonDominatedPaths").Cells(1 * (NumberOfCombinedPaths + 2) +
ComboNumber + 4, 3 + Thread) = ThreadInComboNumber(ComboNumber, Thread)
  If ThreadInComboNumber(ComboNumber, Thread) = 0 Then
ComboNumberNonDominated(ComboNumber) = 0
Next Thread
If ComboNumberNonDominated(ComboNumber) = 1 Then
Worksheets("NonDominatedPaths").Cells(1 * (NumberOfCombinedPaths + 2) +
ComboNumber + 4, 1) = ComboNumber

 If ComboNumberNonDominated(ComboNumber) = 1 And
(MaximumPathDistance(ComboNumber) = NodeValueUpper(EndNode,
WCardinality(EndNode))) Then
```

```
   Worksheets("NonDominatedPaths").Cells(1 * (NumberOfCombinedPaths + 2) +
ComboNumber + 4, 1) = ComboNumber
Else
   ComboNumberNonDominated(ComboNumber) = 0
End If
Worksheets("NonDominatedPaths").Cells(1 * (NumberOfCombinedPaths + 2) +
ComboNumber + 4, 1) = ComboNumberNonDominated(ComboNumber)
Next ComboNumber


End Sub

Sub MinimizeRegret()

'SmallestMinimum = Infinity
SmallestChange = Infinity
'LargestMaximum = 0
For ComboNumber = 1 To NumberOfCombinedPaths
   MinimumPathDistance(ComboNumber) = 0
   MaximumPathDistance(ComboNumber) = 0
   PathNumber = CombinedPath(ComboNumber, 1)
   For Position = LastPathPositionEnd(PathNumber) To 2 Step -1
      ToNode = EndPathNode(PathNumber, Position)
      FromNode = EndPathNode(PathNumber, Position - 1)
      Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1) +
Position, 3) = FromNode
      Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1) +
Position, 4) = ToNode
      'Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1)
+ Position, 9) = NodeValueLower(ToNode, WCardinality(ToNode))
      'Worksheets("CombinedPaths").Cells((MaxPathLength + 2) * (ComboNumber - 1)
+ Position, 10) = NodeValueUpper(ToNode, WCardinality(ToNode))
      MinimumPathDistance(ComboNumber) = MinimumPathDistance(ComboNumber)
+ ArcLower(FromNode, ToNode)
      MaximumPathDistance(ComboNumber) = MaximumPathDistance(ComboNumber)
+ ArcUpper(FromNode, ToNode)
   Next Position
   CumulativeDistanceChange(ComboNumber) =
MaximumPathDistance(ComboNumber) - MinimumPathDistance(ComboNumber)
   'If MinimumPathDistance(ComboNumber) < SmallestMinimum Then
   'SmallestMinimumCombo = ComboNumber
   'SmallestMinimum = MinimumPathDistance(ComboNumber)
   'End If
   'If MaximumPathDistance(ComboNumber) < LargestMaximum Then
   'LargestMaximumCombo = ComboNumber
   'LargestMaximum = MaximumPathDistance(ComboNumber)
```

```
   'End If
   If CumulativeDistanceChange(ComboNumber) < SmallestChange Then
   MinimizedRegretCombo = ComboNumber
   SmallestChange = CumulativeDistanceChange(ComboNumber)
   End If
   Worksheets("NonDominatedPaths").Cells(ComboNumber + 4, 4) =
MinimumPathDistance(ComboNumber)
   Worksheets("NonDominatedPaths").Cells(ComboNumber + 4, 5) =
MaximumPathDistance(ComboNumber)
   Worksheets("NonDominatedPaths").Cells(ComboNumber + 4, 7) =
CumulativeDistanceChange(ComboNumber)
   Worksheets("NonDominatedPaths").Cells(ComboNumber + 4, 1) =
ComboPathPoints(ComboNumber)
Next ComboNumber
Worksheets("NonDominatedPaths").Cells(2, 1) = SmallestChange
Worksheets("NonDominatedPaths").Cells(2, 2) = MinimizedRegretCombo
Worksheets("NonDominatedPaths").Cells(1, 1) = "SmallestChange"
Worksheets("NonDominatedPaths").Cells(1, 2) = "MinimizedRegretCombo"
Worksheets("NonDominatedPaths").Cells(4, 1) = "ComboPoints"


End Sub

Sub GetArcs()

For FromNode = 1 To EndNode
   For ToNode = 1 To EndNode
      If ArcLower(FromNode, ToNode) < Infinity Then
       NumberOfArcs = NumberOfArcs + 1
       RangeCalc = RangeCalc + 1
       OriginatingNode(NumberOfArcs) = FromNode
       TerminatingNode(NumberOfArcs) = ToNode
      End If
   Next ToNode
Next FromNode
'loop = n ^ 2

RangeCalc = 0
For ComboNumber = 1 To NumberOfCombinedPaths
PathNumber = CombinedPath(ComboNumber, 1)

For Position = 1 To LastPathPositionEnd(PathNumber)

ToNode = EndPathNode(PathNumber, Position + 1)
FromNode = EndPathNode(PathNumber, Position)
ToNodeThread = EndPathThread(PathNumber, Position + 1)
```

```
FromNodeThread = EndPathThread(PathNumber, Position)

Worksheets("DominantArcs").Cells(2, 2) = "PathArc"
Worksheets("DominantArcs").Cells(2, 3) = "PathNumber"

For i = 1 To NumberOfArcs
If (OriginatingNode(i) = FromNode) And (TerminatingNode(i) = ToNode) Then
   PathArc(ComboNumber, Position) = i
   RangeCalc = RangeCalc + 1
   Worksheets("DominantArcs").Cells(2 + RangeCalc, 2) = PathArc(ComboNumber,
Position)
   Worksheets("DominantArcs").Cells(2 + RangeCalc, 3) = ComboNumber
   'Worksheets("DominantArcs").Cells(2 + RangeCalc, 30) =
PathLowArcValue(FromNode, ToNode, PathNumber, Position)
   'Worksheets("DominantArcs").Cells(2 + RangeCalc, 31) =
PathUpperArcValue(FromNode, ToNode, PathNumber, Position)
End If
Next i

Next Position
Next ComboNumber

End Sub

Sub EvaluateDominantArcs()

For ComboNumber = 1 To NumberOfCombinedPaths
PathNumber = CombinedPath(ComboNumber, 1)
For Position = 1 To (LastPathPositionEnd(PathNumber) - 1)
For i = 1 To NumberOfArcs
   If PathArc(ComboNumber, Position) = i Then NumberOfPathsContainingArc(i) =
NumberOfPathsContainingArc(i) + 1
Next i
Next Position
'Next PathNumber
Next ComboNumber

ArcsInShortPath = 0
For i = 1 To NumberOfArcs
   Worksheets("DominantArcs").Cells(2 + i, 6) = i
   Worksheets("DominantArcs").Cells(2 + i, 5) = NumberOfPathsContainingArc(i)
   ArcsInShortPath = ArcsInShortPath + NumberOfPathsContainingArc(i)
Next i

'For i = 1 To NumberOfArcs
'   RelativeWorthArc(i) = NumberOfPathsContainingArc(i) / ArcsInShortPath
```

```
'   Worksheets("DominantArcs").Cells(2 + i, 7) = RelativeWorthArc(i)
'Next i

Worksheets("DominantArcs").Cells(2, 7) = "RelativeWorthArc"
Worksheets("DominantArcs").Cells(2, 6) = "ArcNumber"
Worksheets("DominantArcs").Cells(2, 5) = "NumberOfPathsContainingArc"
Worksheets("DominantArcs").Cells(2, 10) = "PathNumber"
Worksheets("DominantArcs").Cells(2, 11) = "PathPoints"

For ComboNumber = 1 To NumberOfCombinedPaths
PathNumber = CombinedPath(ComboNumber, 1)
'For PathNumber = 1 To TotalNumberOfPaths
PathPoints(ComboNumber) = 0
For Position = 1 To (LastPathPositionEnd(PathNumber) - 1)
   PathPoints(ComboNumber) = PathPoints(ComboNumber) +
NumberOfPathsContainingArc(PathArc(ComboNumber, Position))
Next Position

PathPoints(ComboNumber) = PathPoints(ComboNumber) /
(LastPathPositionEnd(CombinedPath(ComboNumber, 1)) - 1)
Worksheets("DominantArcs").Cells(2 + ComboNumber, 10) = ComboNumber
Worksheets("DominantArcs").Cells(2 + ComboNumber, 11) =
PathPoints(ComboNumber)
'Next PathNumber
Next ComboNumber

End Sub

Sub WriteArcInfo()
RangeCalc = 0
For i = 1 To NumberOfArcs
RangeCalc = RangeCalc + 1
For PathNumber = 1 To TotalNumberOfPaths
For Position = LastPathPositionEnd(PathNumber) To 2 Step -1

ToNode = EndPathNode(PathNumber, Position)
FromNode = EndPathNode(PathNumber, Position - 1)
ToNodeThread = EndPathThread(PathNumber, Position)
FromNodeThread = EndPathThread(PathNumber, Position - 1)
'For i = 1 To NumberOfArcs
If (PathArc(PathNumber, Position - 1) = i) Then
   RangeCalc = RangeCalc + 1
   Worksheets("ArcInfo").Cells(2 + RangeCalc, 2) = i
   Worksheets("ArcInfo").Cells(2 + RangeCalc, 3) = PathNumber
   Worksheets("ArcInfo").Cells(2 + RangeCalc, 5) = PathLowArcValue(FromNode,
ToNode, PathNumber, Position)
```

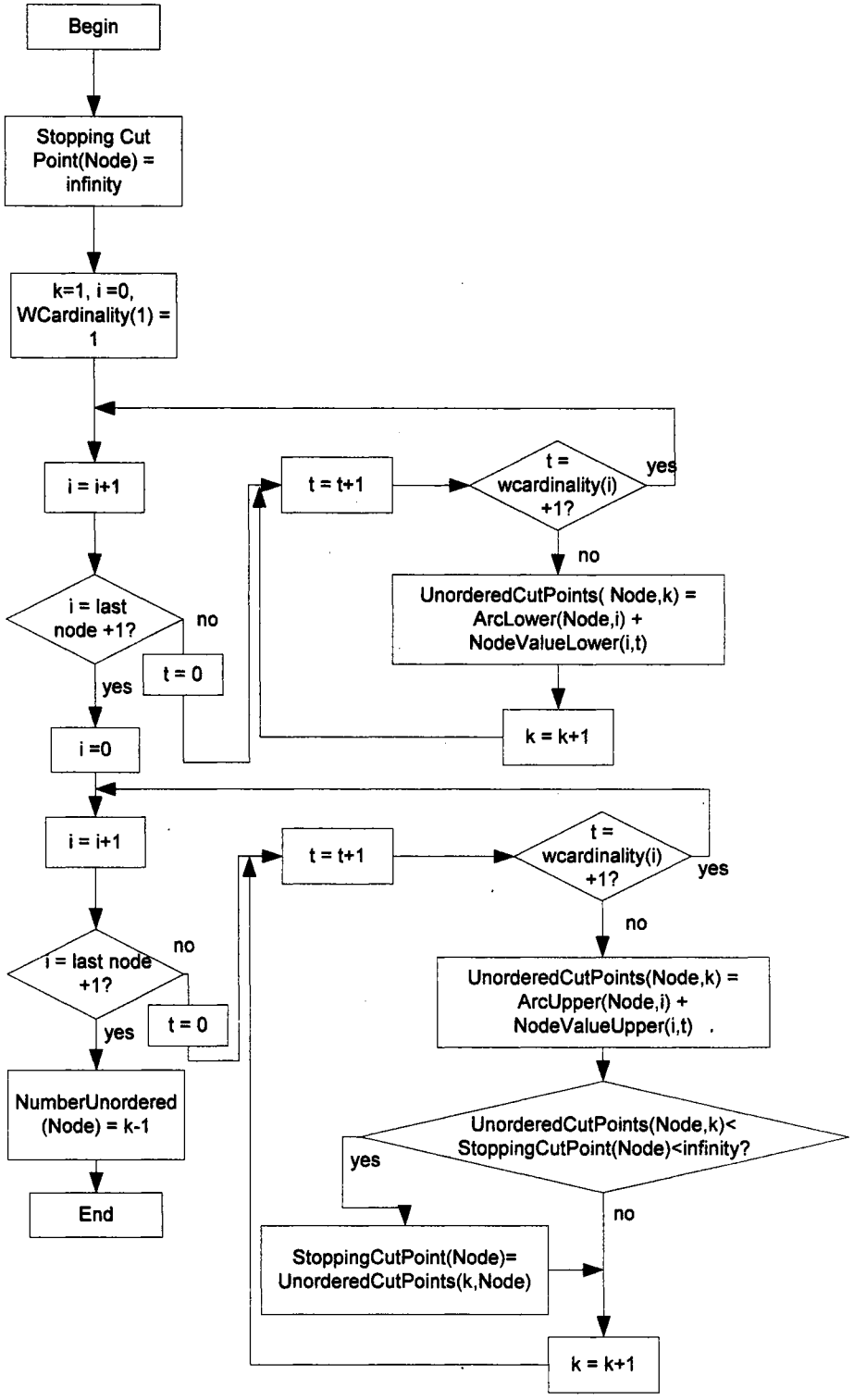
```
   Worksheets("ArcInfo").Cells(2 + RangeCalc, 6) = PathUpperArcValue(FromNode,
ToNode, PathNumber, Position)
   Worksheets("ArcInfo").Cells(2 + RangeCalc, 11) = OriginatingNode(i)
   Worksheets("ArcInfo").Cells(2 + RangeCalc, 12) = TerminatingNode(i)
   Worksheets("ArcInfo").Cells(2 + RangeCalc, 8) = ArcLower(OriginatingNode(i),
TerminatingNode(i))
   Worksheets("ArcInfo").Cells(2 + RangeCalc, 9) = ArcUpper(OriginatingNode(i),
TerminatingNode(i))
End If
'Next i

Next Position
Next PathNumber
Next i

End Sub

Sub FindSubPaths()

RangeCalc = 1
For SubPathLength = 3 To MaxPathLength
Number = 1
For ComboNumber = 1 To NumberOfCombinedPaths
   PathNumber = CombinedPath(ComboNumber, 1)
   For StartingSet = 1 To (LastPathPositionEnd(PathNumber) - SubPathLength)
      SubPathComboNumber(SubPathLength, Number) = ComboNumber
      Worksheets("SubPaths").Cells(RangeCalc + 2, 4) = Number
      For Position = 1 To SubPathLength
         SubPath(SubPathLength, Number, Position) = EndPathNode(PathNumber,
StartingSet + Position)
         Worksheets("SubPaths").Cells(RangeCalc + 2, 2) = SubPath(SubPathLength,
Number, Position)
         RangeCalc = RangeCalc + 1
      Next Position
         Worksheets("SubPaths").Cells(RangeCalc + 1, 3) =
SubPathComboNumber(SubPathLength, Number)
         Number = Number + 1
   RangeCalc = RangeCalc + 1
   Next StartingSet

Next ComboNumber
NumberOfSubPaths(SubPathLength) = Number - 1
Worksheets("SubPaths").Cells(RangeCalc + 2, 5) = NumberOfSubPaths(SubPathLength)
Next SubPathLength

End Sub
```

```
Sub CheckIdenticalSubPaths()

'Initialize Match
For SubPathLength = 3 To MaxPathLength
For Number = 1 To NumberOfSubPaths(SubPathLength)
For j = 1 To NumberOfSubPaths(SubPathLength)
   SubPathMatch(SubPathLength, Number, j) = 1
   ComboSubPathMatch(SubPathLength, Number, j) = 1
Next j
Next Number
Next SubPathLength

RangeCalc = 1
For SubPathLength = 3 To MaxPathLength
RangeCalc = RangeCalc + 1
For Number = 1 To NumberOfSubPaths(SubPathLength)
NumberSubMatches(SubPathLength, Number) = 1
RangeCalc = RangeCalc + 1
For j = Number To NumberOfSubPaths(SubPathLength)
   For Position = 1 To SubPathLength
      If SubPath(SubPathLength, Number, Position) = SubPath(SubPathLength, j,
Position) And (SubPathMatch(SubPathLength, Number, j) = 1) Then
         SubPathMatch(SubPathLength, Number, j) = 1
         Else: SubPathMatch(SubPathLength, Number, j) = 0
      End If
   Next Position
Next j
Next Number
Next SubPathLength
End Sub

Sub FindCombosInSubPaths()

For SubPathLength = 3 To MaxPathLength
For Number = 1 To NumberOfSubPaths(SubPathLength)
   ComboSubPathFound(SubPathLength, Number) = 0
   NumberSubMatches(SubPathLength, Number) = 1
Next Number

For Number = 1 To NumberOfSubPaths(SubPathLength)
   If ComboSubPathFound(SubPathLength, Number) = 0 Then
      For j = (Number + 1) To NumberOfSubPaths(SubPathLength)
         If (SubPathMatch(SubPathLength, Number, j) = 1) Then
            ComboSubPathFound(SubPathLength, Number) = 1
            ComboSubPathFound(SubPathLength, j) = 1
```

```
            NumberSubMatches(SubPathLength, Number) =
NumberSubMatches(SubPathLength, Number) + 1

          End If
        Next j
    End If
Next Number

Next SubPathLength

RangeCalc = 1
Worksheets("SubPaths").Cells(2, 12) = "SubPathLength"
Worksheets("SubPaths").Cells(2, 14) = "NumberSubMatches"
For SubPathLength = 3 To MaxPathLength
    Range = 0
    RangeCalc = RangeCalc + 1
    For Number = 1 To NumberOfSubPaths(SubPathLength)
    If NumberSubMatches(SubPathLength, Number) > 1 Then
        RangeCalc = RangeCalc + 1
        Worksheets("SubPaths").Cells(RangeCalc + SubPathLength, 12) = SubPathLength
        Worksheets("SubPaths").Cells(RangeCalc + SubPathLength, 14) =
NumberSubMatches(SubPathLength, Number)
        'Worksheets("SubPaths").Cells(RangeCalc + 2, 15) = Number
    For Position = 1 To SubPathLength
     Worksheets("SubPaths").Cells(RangeCalc + SubPathLength, 16 + Position) =
SubPath(SubPathLength, Number, Position)
    Next Position
    Worksheets("SubPaths").Cells(RangeCalc + SubPathLength, 17 + Position) =
SubPathComboNumber(SubPathLength, Number)
    For j = (Number + 1) To NumberOfSubPaths(SubPathLength)

        If (SubPathMatch(SubPathLength, Number, j) = 1) Then
          Range = Range + 1
          Worksheets("SubPaths").Cells(RangeCalc + SubPathLength, 17 + Position +
Range) = SubPathComboNumber(SubPathLength, j)
        End If
    Next j
End If

Next Number
Next SubPathLength

End Sub
```

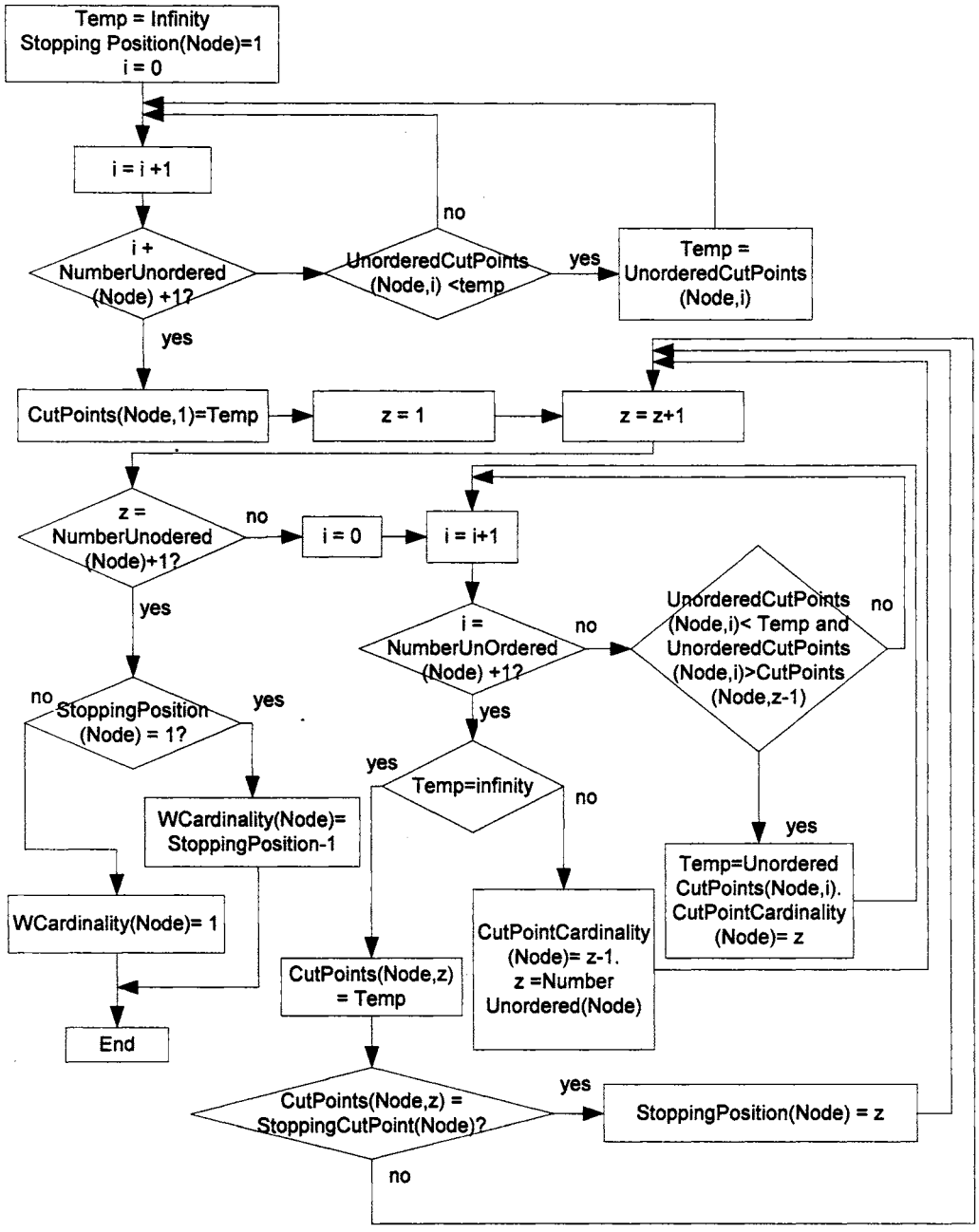
APPENDIX B

FLOW CHART IDENTIFY CUT POINTS



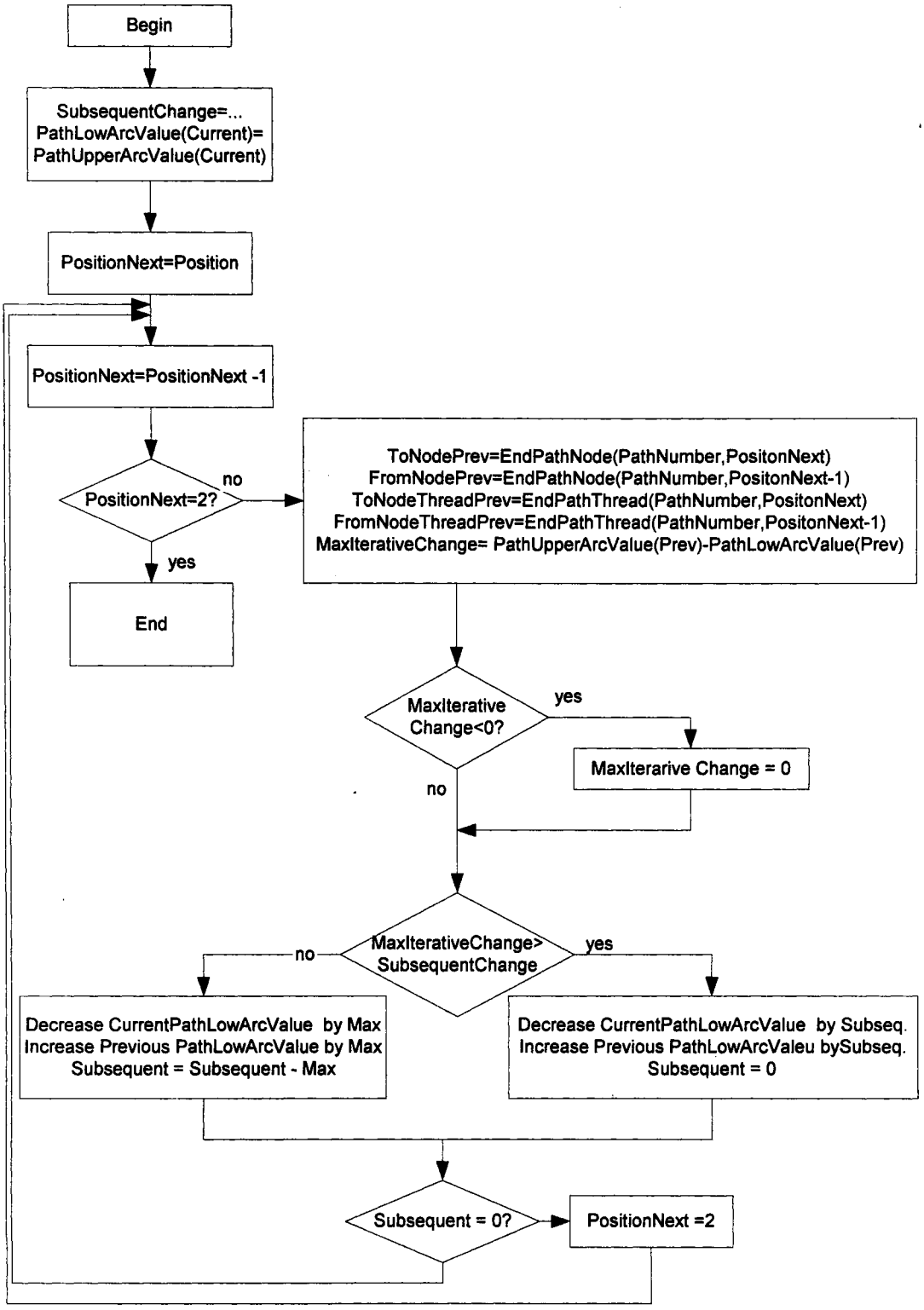
APPENDIX C

FLOW CHART ORDER CUT POINTS



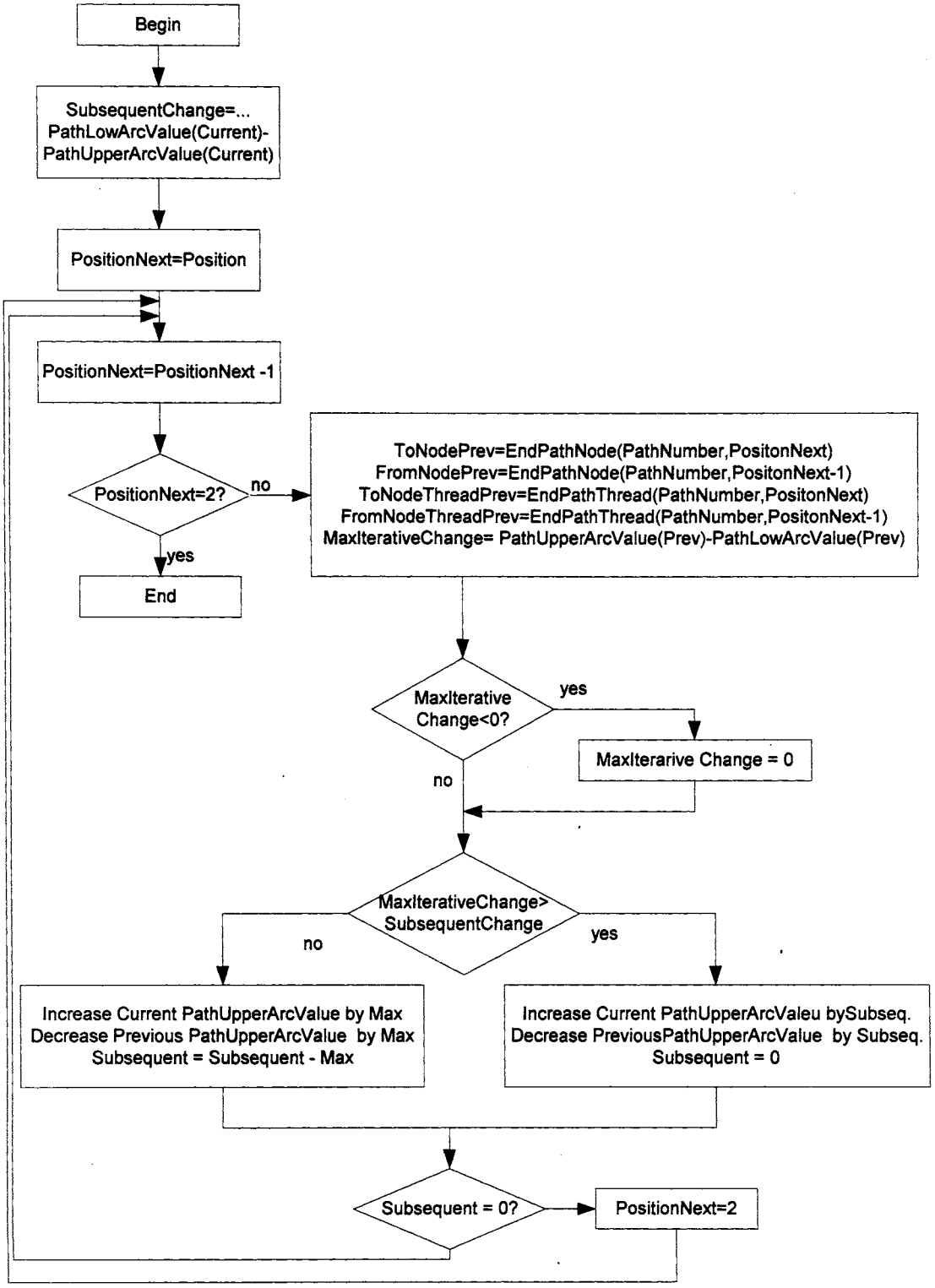
APPENDIX D

FLOW CHART DECREASE PATH LOW ARC VALUE



APPENDIX E

FLOW CHART INCREASE PATH HIGH ARC VALUE



APPENDIX F

SHORTEST PATHS 10-NODE NETWORK IV

PATH NUMBER	NODE 1	NODE 2	NODE 3	NODE 4	NODE 5	NODE 6
1	101	301	601	1001		
2	101	301	601	1002		
3	101	401	701	801	1002	
4	101	301	601	1003		
5	101	401	701	1003		
6	101	401	701	801	1003	
7	101	301	601	1004		
8	101	301	602	1004		
9	101	401	602	1004		
10	101	401	701	1004		
11	101	401	701	801	1004	
12	101	301	601	1005		
13	101	301	602	1005		
14	101	401	602	1005		
15	101	401	701	1005		
16	101	401	701	801	1005	
17	101	401	701	802	1005	
18	101	401	702	802	1005	
19	101	201	501	702	802	1005
20	101	301	601	1006		
21	101	301	602	1006		
22	101	401	602	1006		
23	101	401	701	1006		
24	101	401	701	801	1006	
25	101	401	701	802	1006	
26	101	401	702	802	1006	
27	101	201	501	702	802	1006
28	101	201	501	803	1006	
29	101	401	701	803	1006	
30	101	401	702	803	1006	
31	101	201	501	702	803	1006
32	101	401	703	803	1006	
33	101	402	703	803	1006	
34	101	301	402	703	803	1006
35	101	201	501	703	803	1006
36	101	301	601	1007		
37	101	301	602	1007		
38	101	401	602	1007		
39	101	401	701	1007		
40	101	401	701	802	1007	
41	101	401	702	802	1007	
42	101	201	501	702	802	1007
43	101	201	501	803	1007	
44	101	401	701	803	1007	
45	101	401	702	803	1007	
46	101	201	501	702	803	1007
47	101	401	703	803	1007	
48	101	402	703	803	1007	
49	101	301	402	703	803	1007
50	101	201	501	703	803	1007

PATH NUMBER	NODE 1	NODE 2	NODE 3	NODE 4	NODE 5	NODE 6
270	101	402	704	1015		
271	101	301	402	704	1015	
272	101	201	501	704	1015	
273	101	201	501	806	1015	
274	101	401	703	806	1015	
275	101	402	703	806	1015	
276	101	301	402	703	806	1015
277	101	201	501	703	806	1015
278	101	402	704	806	1015	
279	101	301	402	704	806	1015
280	101	201	501	704	806	1015
281	101	402	705	806	1015	
282	101	301	402	705	806	1015
283	101	201	501	705	806	1015
284	101	301	601	705	806	1015
285	101	301	603	1016		
286	101	401	603	1016		
287	101	402	603	1016		
288	101	301	402	603	1016	
289	101	401	701	1016	20	
290	101	401	702	1016	20	
291	101	201	501	702	1016	
292	101	401	703	1016	40	
293	101	402	703	1016		
294	101	301	402	703	1016	
295	101	201	501	703	1016	
296	101	402	704	1016	20	
297	101	301	402	704	1016	
298	101	201	501	704	1016	
299	101	402	705	1016	20	
300	101	301	402	705	1016	
301	101	201	501	705	1016	
302	101	301	601	705	1016	
303	101	201	501	806	1016	
304	101	401	703	806	1016	
305	101	402	703	806	1016	
306	101	301	402	703	806	1016
307	101	201	501	703	806	1016
308	101	402	704	806	1016	
309	101	301	402	704	806	1016
310	101	201	501	704	806	1016
311	101	402	705	806	1016	
312	101	301	402	705	806	1016
313	101	201	501	705	806	1016
314	101	301	601	705	806	1016
315	101	301	603	1017		
316	101	401	603	1017		
317	101	402	603	1017		
318	101	301	402	603	1017	
319	101	401	701	1017		

PATH NUMBER	NODE 1	NODE 2	NODE 3	NODE 4	NODE 5	NODE 6
51	101	301	602	1008		
52	101	401	602	1008		
53	101	401	701	1008		
54	101	201	501	803	1008	
55	101	401	701	803	1008	
56	101	401	702	803	1008	
57	101	201	501	702	803	1008
58	101	401	703	803	1008	
59	101	402	703	803	1008	
60	101	301	402	703	803	1008
61	101	201	501	703	803	1008
62	101	301	602	1009		
63	101	401	602	1009		
64	101	401	701	1009		
65	101	401	702	1009		
66	101	201	501	702	1009	
67	101	201	501	803	1009	
68	101	401	701	803	1009	
69	101	401	702	803	1009	
70	101	201	501	702	803	1009
71	101	401	703	803	1009	
72	101	402	703	803	1009	
73	101	301	402	703	803	1009
74	101	201	501	703	803	1009
75	101	301	602	1010		
76	101	401	602	1010		
77	101	401	701	1010		
78	101	401	702	1010		
79	101	201	501	702	1010	
80	101	401	703	1010		
81	101	402	703	1010		
82	101	301	402	703	1010	
83	101	201	501	703	1010	
84	101	201	501	803	1010	
85	101	401	701	803	1010	
86	101	401	702	803	1010	
87	101	201	501	702	803	1010
88	101	401	703	803	1010	
89	101	402	703	803	1010	
90	101	301	402	703	803	1010
91	101	201	501	703	803	1010
92	101	301	602	1011		
93	101	401	602	1011		
94	101	301	603	1011		
95	101	401	603	1011		
96	101	402	603	1011		
97	101	301	402	603	1011	
98	101	401	701	1011		
99	101	401	702	1011		
100	101	201	501	702	1011	
101	101	401	703	1011		
102	101	402	703	1011		
103	101	301	402	703	1011	
104	101	201	501	703	1011	
105	101	201	501	803	1011	
106	101	401	701	803	1011	
107	101	401	702	803	1011	
108	101	201	501	702	803	1011
109	101	401	703	803	1011	
110	101	402	703	803	1011	
320	101	401	702	1017		
321	101	201	501	702	1017	
322	101	401	703	1017		
323	101	402	703	1017		
324	101	301	402	703	1017	
325	101	201	501	703	1017	
326	101	402	704	1017		
327	101	301	402	704	1017	
328	101	201	501	704	1017	
329	101	402	705	1017		
330	101	301	402	705	1017	
331	101	201	501	705	1017	
332	101	301	601	705	1017	
333	101	201	501	806	1017	
334	101	401	703	806	1017	
335	101	402	703	806	1017	
336	101	301	402	703	806	1017
337	101	201	501	703	806	1017
338	101	402	704	806	1017	
339	101	301	402	704	806	1017
340	101	201	501	704	806	1017
341	101	402	705	806	1017	
342	101	301	402	705	806	1017
343	101	201	501	705	806	1017
344	101	301	601	705	806	1017
345	101	201	501	807	1017	
346	101	401	703	807	1017	
347	101	402	703	807	1017	
348	101	301	402	703	807	1017
349	101	201	501	703	807	1017
350	101	402	704	807	1017	
351	101	301	402	704	807	1017
352	101	201	501	704	807	1017
353	101	402	705	807	1017	
354	101	301	402	705	807	1017
355	101	201	501	705	807	1017
356	101	301	601	705	807	1017
357	101	402	706	807	1017	
358	101	301	402	706	807	1017
359	101	201	501	706	807	1017
360	101	201	502	706	807	1017
361	101	401	502	706	807	1017
362	101	301	601	706	807	1017
363	101	301	603	1018		
364	101	401	603	1018		
365	101	402	603	1018		
366	101	301	402	603	1018	
367	101	401	701	1018		
368	101	401	702	1018		
369	101	201	501	702	1018	
370	101	401	703	1018		
371	101	402	703	1018		
372	101	301	402	703	1018	
373	101	201	501	703	1018	
374	101	402	704	1018		
375	101	301	402	704	1018	
376	101	201	501	704	1018	
377	101	402	705	1018		
378	101	301	402	705	1018	
379	101	201	501	705	1018	

PATH NUMBER	NODE 1	NODE 2	NODE 3	NODE 4	NODE 5	NODE 6
111	101	301	402	703	803	1011
112	101	201	501	703	803	1011
113	101	201	501	804	1011	
114	101	401	701	804	1011	
115	101	401	702	804	1011	
116	101	201	501	702	804	1011
117	101	401	703	804	1011	
118	101	402	703	804	1011	
119	101	301	402	703	804	1011
120	101	201	501	703	804	1011
121	101	402	704	804	1011	
122	101	301	402	704	804	1011
123	101	201	501	704	804	1011
124	101	301	602	1012		
125	101	401	602	1012		
126	101	301	603	1012		
127	101	401	603	1012		
128	101	402	603	1012		
129	101	301	402	603	1012	
130	101	401	701	1012		
131	101	401	702	1012		
132	101	201	501	702	1012	
133	101	401	703	1012		
134	101	402	703	1012		
135	101	301	402	703	1012	
136	101	201	501	703	1012	
137	101	201	501	803	1012	
138	101	401	701	803	1012	
139	101	401	702	803	1012	
140	101	201	501	702	803	1012
141	101	401	703	803	1012	
142	101	402	703	803	1012	
143	101	301	402	703	803	1012
144	101	201	501	703	803	1012
145	101	201	501	804	1012	
146	101	401	701	804	1012	
147	101	401	702	804	1012	
148	101	201	501	702	804	1012
149	101	401	703	804	1012	
150	101	402	703	804	1012	
151	101	301	402	703	804	1012
152	101	201	501	703	804	1012
153	101	402	704	804	1012	
154	101	301	402	704	804	1012
155	101	201	501	704	804	1012
156	101	201	501	805	1012	
157	101	401	702	805	1012	
158	101	201	501	702	805	1012
159	101	401	703	805	1012	
160	101	402	703	805	1012	
161	101	301	402	703	805	1012
162	101	201	501	703	805	1012
163	101	402	704	805	1012	
164	101	301	402	704	805	1012
165	101	201	501	704	805	1012
166	101	402	705	805	1012	
167	101	301	402	705	805	1012
168	101	201	501	705	805	1012
169	101	301	601	705	805	1012
170	101	301	602	1013		

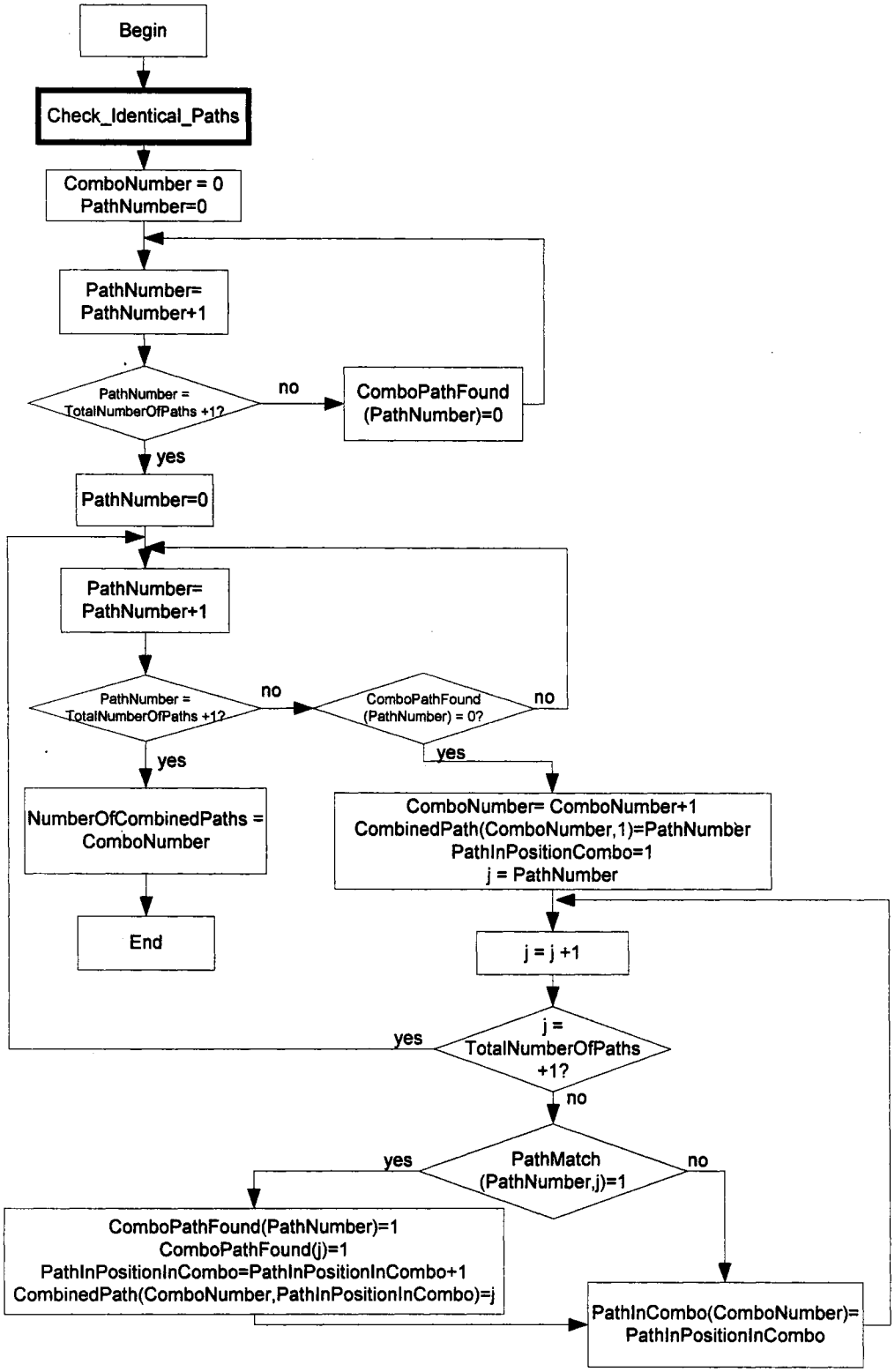
PATH NUMBER	NODE 1	NODE 2	NODE 3	NODE 4	NODE 5	NODE 6
380	101	301	601	705	1018	
381	101	201	501	806	1018	
382	101	401	703	806	1018	
383	101	402	703	806	1018	
384	101	301	402	703	806	1018
385	101	201	501	703	806	1018
386	101	402	704	806	1018	
387	101	301	402	704	806	1018
388	101	201	501	704	806	1018
389	101	402	705	806	1018	
390	101	301	402	705	806	1018
391	101	201	501	705	806	1018
392	101	301	601	705	806	1018
393	101	201	501	807	1018	
394	101	401	703	807	1018	
395	101	402	703	807	1018	
396	101	301	402	703	807	1018
397	101	201	501	703	807	1018
398	101	402	704	807	1018	
399	101	301	402	704	807	1018
400	101	201	501	704	807	1018
401	101	402	705	807	1018	
402	101	301	402	705	807	1018
403	101	201	501	705	807	1018
404	101	301	601	705	807	1018
405	101	402	706	807	1018	
406	101	301	402	706	807	1018
407	101	201	501	706	807	1018
408	101	201	502	706	807	1018
409	101	401	502	706	807	1018
410	101	301	601	706	807	1018
411	101	201	501	808	1018	
412	101	201	502	808	1018	
413	101	401	502	808	1018	
414	101	402	704	808	1018	
415	101	301	402	704	808	1018
416	101	201	501	704	808	1018
417	101	402	705	808	1018	
418	101	301	402	705	808	1018
419	101	201	501	705	808	1018
420	101	301	601	705	808	1018
421	101	402	706	808	1018	
422	101	301	402	706	808	1018
423	101	201	501	706	808	1018
424	101	201	502	706	808	1018
425	101	401	502	706	808	1018
426	101	301	601	706	808	1018
427	101	301	603	1019		
428	101	401	603	1019		
429	101	402	603	1019		
430	101	301	402	603	1019	
431	101	401	701	1019		
432	101	401	702	1019		
433	101	201	501	702	1019	
434	101	401	703	1019		
435	101	402	703	1019		
436	101	301	402	703	1019	
437	101	201	501	703	1019	
438	101	402	704	1019		
439	101	301	402	704	1019	

PATH NUMBER	NODE 1	NODE 2	NODE 3	NODE 4	NODE 5	NODE 6
171	101	401	602	1013		
172	101	301	603	1013		
173	101	401	603	1013		
174	101	402	603	1013		
175	101	301	402	603	1013	
176	101	401	701	1013		
177	101	401	702	1013		
178	101	201	501	702	1013	
179	101	401	703	1013		
180	101	402	703	1013		
181	101	301	402	703	1013	
182	101	201	501	703	1013	
183	101	201	501	804	1013	
184	101	401	701	804	1013	
185	101	401	702	804	1013	
186	101	201	501	702	804	1013
187	101	401	703	804	1013	
188	101	402	703	804	1013	
189	101	301	402	703	804	1013
190	101	201	501	703	804	1013
191	101	402	704	804	1013	
192	101	301	402	704	804	1013
193	101	201	501	704	804	1013
194	101	201	501	805	1013	
195	101	401	702	805	1013	
196	101	201	501	702	805	1013
197	101	401	703	805	1013	
198	101	402	703	805	1013	
199	101	301	402	703	805	1013
200	101	201	501	703	805	1013
201	101	402	704	805	1013	
202	101	301	402	704	805	1013
203	101	201	501	704	805	1013
204	101	402	705	805	1013	
205	101	301	402	705	805	1013
206	101	201	501	705	805	1013
207	101	301	601	705	805	1013
208	101	201	501	806	1013	
209	101	401	703	806	1013	
210	101	402	703	806	1013	
211	101	301	402	703	806	1013
212	101	201	501	703	806	1013
213	101	402	704	806	1013	
214	101	301	402	704	806	1013
215	101	201	501	704	806	1013
216	101	402	705	806	1013	
217	101	301	402	705	806	1013
218	101	201	501	705	806	1013
219	101	301	601	705	806	1013
220	101	301	602	1014		
221	101	401	602	1014		
222	101	301	603	1014		
223	101	401	603	1014		
224	101	402	603	1014		
225	101	301	402	603	1014	
226	101	401	701	1014		
227	101	401	702	1014		
228	101	201	501	702	1014	
229	101	401	703	1014		
230	101	402	703	1014		
440	101	201	501	704	1019	
441	101	402	705	1019		
442	101	301	402	705	1019	
443	101	201	501	705	1019	
444	101	301	601	705	1019	
445	101	201	501	807	1019	
446	101	401	703	807	1019	
447	101	402	703	807	1019	
448	101	301	402	703	807	1019
449	101	201	501	703	807	1019
450	101	402	704	807	1019	
451	101	301	402	704	807	1019
452	101	201	501	704	807	1019
453	101	402	705	807	1019	
454	101	301	402	705	807	1019
455	101	201	501	705	807	1019
456	101	301	601	705	807	1019
457	101	402	706	807	1019	
458	101	301	402	706	807	1019
459	101	201	501	706	807	1019
460	101	201	502	706	807	1019
461	101	401	502	706	807	1019
462	101	301	601	706	807	1019
463	101	201	501	808	1019	
464	101	201	502	808	1019	
465	101	401	502	808	1019	
466	101	402	704	808	1019	
467	101	301	402	704	808	1019
468	101	201	501	704	808	1019
469	101	402	705	808	1019	
470	101	301	402	705	808	1019
471	101	201	501	705	808	1019
472	101	301	601	705	808	1019
473	101	402	706	808	1019	
474	101	301	402	706	808	1019
475	101	201	501	706	808	1019
476	101	201	502	706	808	1019
477	101	401	502	706	808	1019
478	101	301	601	706	808	1019
479	101	201	501	809	1019	
480	101	201	502	809	1019	
481	101	401	502	809	1019	
482	101	402	705	809	1019	
483	101	301	402	705	809	1019
484	101	201	501	705	809	1019
485	101	301	601	705	809	1019
486	101	402	706	809	1019	
487	101	301	402	706	809	1019
488	101	201	501	706	809	1019
489	101	201	502	706	809	1019
490	101	401	502	706	809	1019
491	101	301	601	706	809	1019
492	101	301	603	1020		
493	101	401	603	1020		
494	101	402	603	1020		
495	101	301	402	603	1020	
496	101	401	701	1020		
497	101	401	702	1020		
498	101	201	501	702	1020	
499	101	401	703	1020		

PATH NUMBER	NODE 1	NODE 2	NODE 3	NODE 4	NODE 5	NODE 6
231	101	301	402	703	1014	
232	101	201	501	703	1014	
233	101	201	501	805	1014	
234	101	401	702	805	1014	
235	101	201	501	702	805	1014
236	101	401	703	805	1014	
237	101	402	703	805	1014	
238	101	301	402	703	805	1014
239	101	201	501	703	805	1014
240	101	402	704	805	1014	
241	101	301	402	704	805	1014
242	101	201	501	704	805	1014
243	101	402	705	805	1014	
244	101	301	402	705	805	1014
245	101	201	501	705	805	1014
246	101	301	601	705	805	1014
247	101	201	501	806	1014	
248	101	401	703	806	1014	
249	101	402	703	806	1014	
250	101	301	402	703	806	1014
251	101	201	501	703	806	1014
252	101	402	704	806	1014	
253	101	301	402	704	806	1014
254	101	201	501	704	806	1014
255	101	402	705	806	1014	
256	101	301	402	705	806	1014
257	101	201	501	705	806	1014
258	101	301	601	705	806	1014
259	101	301	603	1015		
260	101	401	603	1015		
261	101	402	603	1015		
262	101	301	402	603	1015	
263	101	401	701	1015		
264	101	401	702	1015		
265	101	201	501	702	1015	
266	101	401	703	1015		
267	101	402	703	1015		
268	101	301	402	703	1015	
269	101	201	501	703	1015	
500	101	402	703	1020		
501	101	301	402	703	1020	
502	101	201	501	703	1020	
503	101	402	704	1020		
504	101	301	402	704	1020	
505	101	201	501	704	1020	
506	101	402	705	1020		
507	101	301	402	705	1020	
508	101	201	501	705	1020	
509	101	301	601	705	1020	
510	101	201	501	808	1020	
511	101	201	502	808	1020	
512	101	401	502	808	1020	
513	101	402	704	808	1020	
514	101	301	402	704	808	1020
515	101	201	501	704	808	1020
516	101	402	705	808	1020	
517	101	301	402	705	808	1020
518	101	201	501	705	808	1020
519	101	301	601	705	808	1020
520	101	402	706	808	1020	
521	101	301	402	706	808	1020
522	101	201	501	706	808	1020
523	101	201	502	706	808	1020
524	101	401	502	706	808	1020
525	101	301	601	706	808	1020
526	101	201	501	809	1020	
527	101	201	502	809	1020	
528	101	401	502	809	1020	
529	101	402	705	809	1020	
530	101	301	402	705	809	1020
531	101	201	501	705	809	1020
532	101	301	601	705	809	1020
533	101	402	706	809	1020	
534	101	301	402	706	809	1020
535	101	201	501	706	809	1020
536	101	201	502	706	809	1020
537	101	401	502	706	809	1020
538	101	301	601	706	809	1020

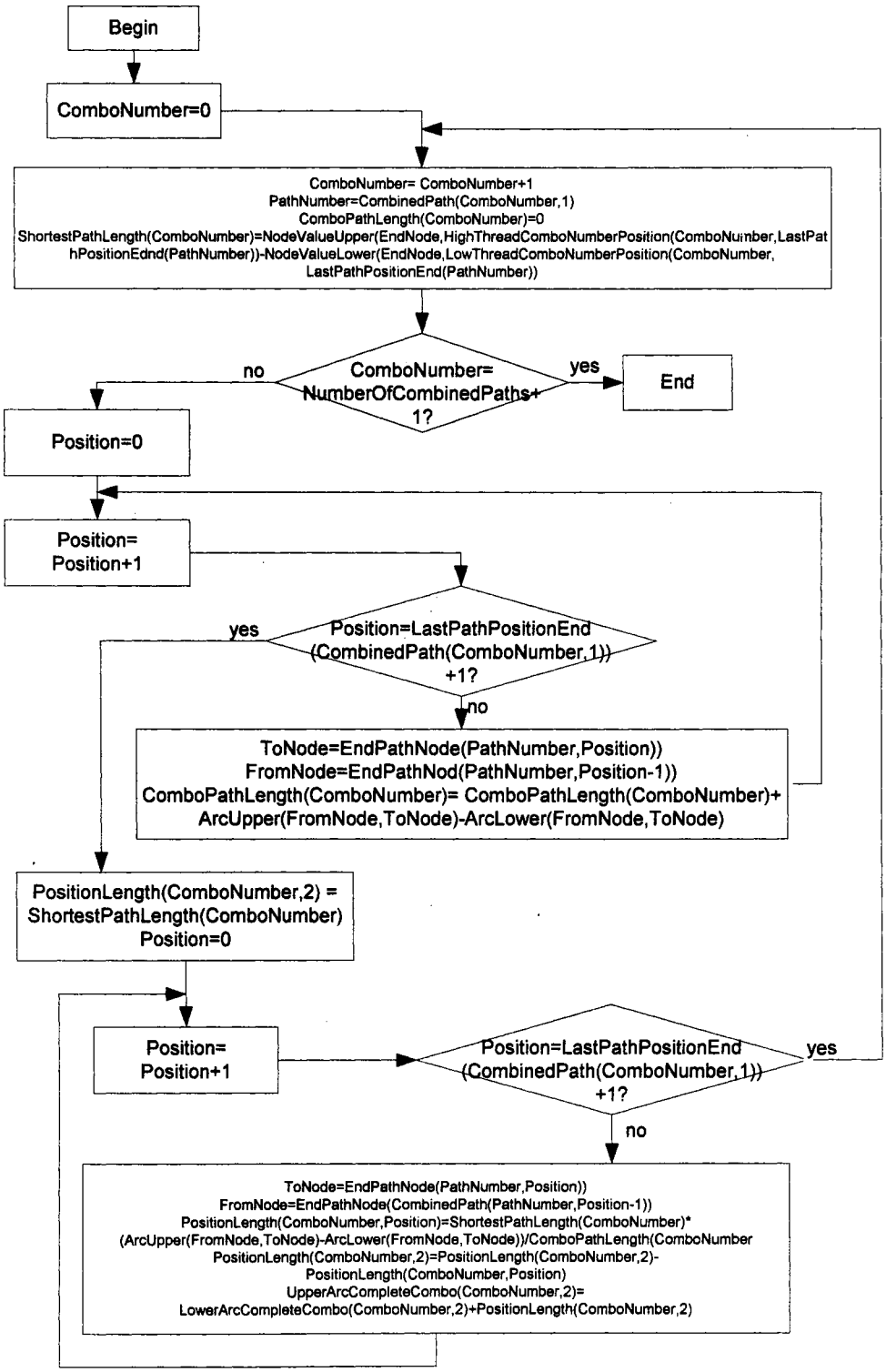
APPENDIX G

FLOW CHART COMBINE PATH THREADS



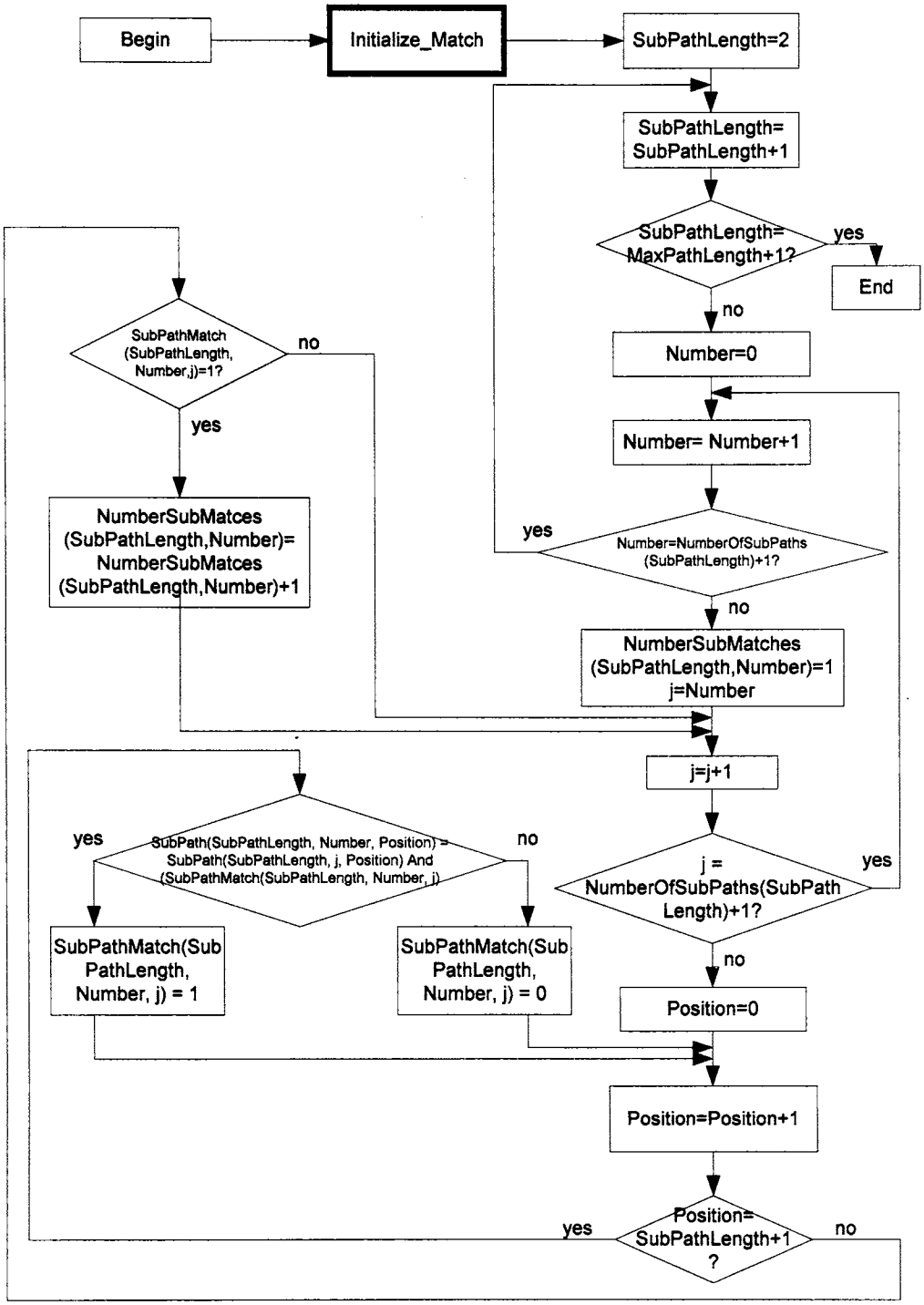
APPENDIX H

FLOW CHART FIND VALUES ON COMBINED PATH



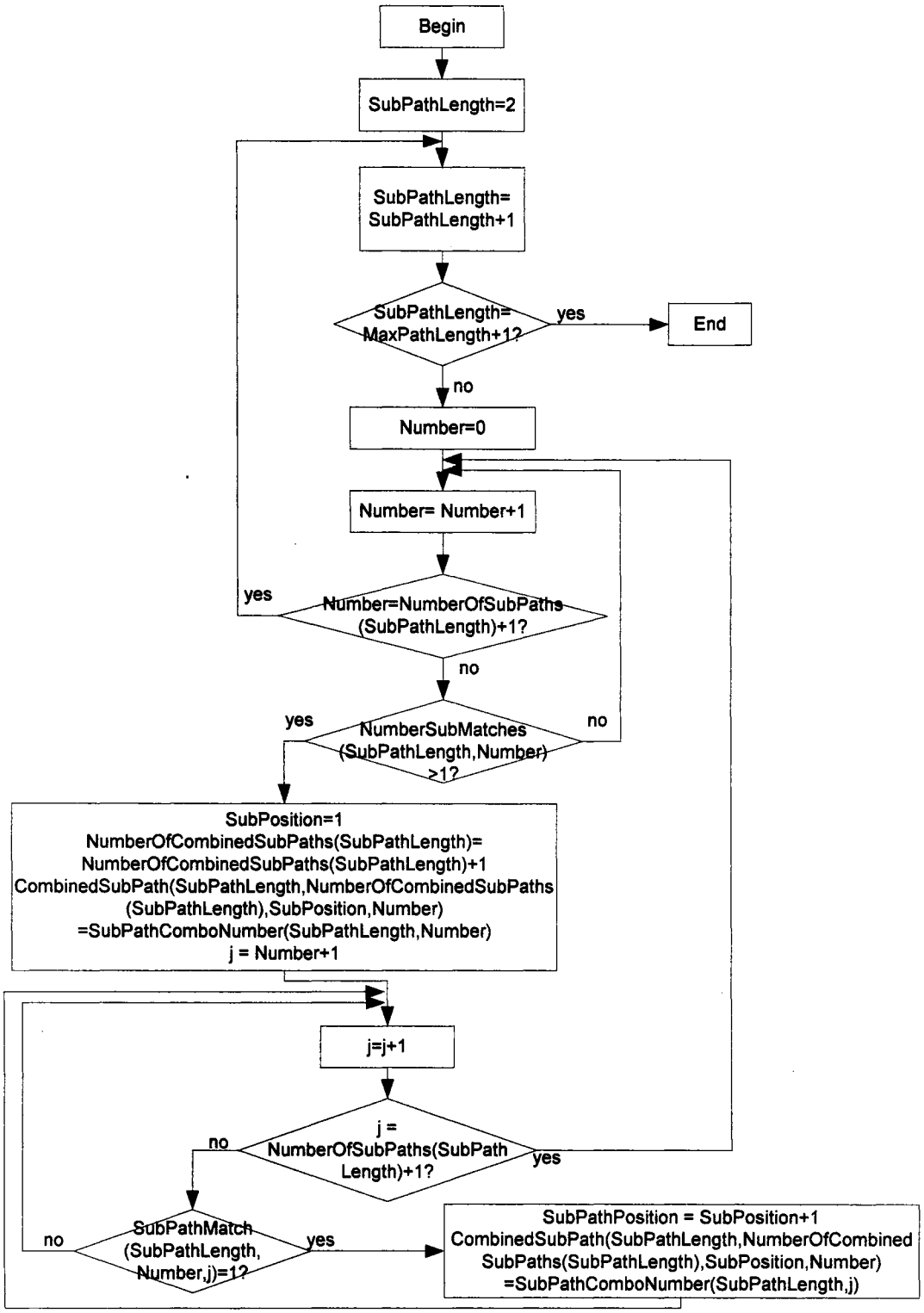
APPENDIX I

FLOW CHART CHECK IDENTICAL SUB-PATHS



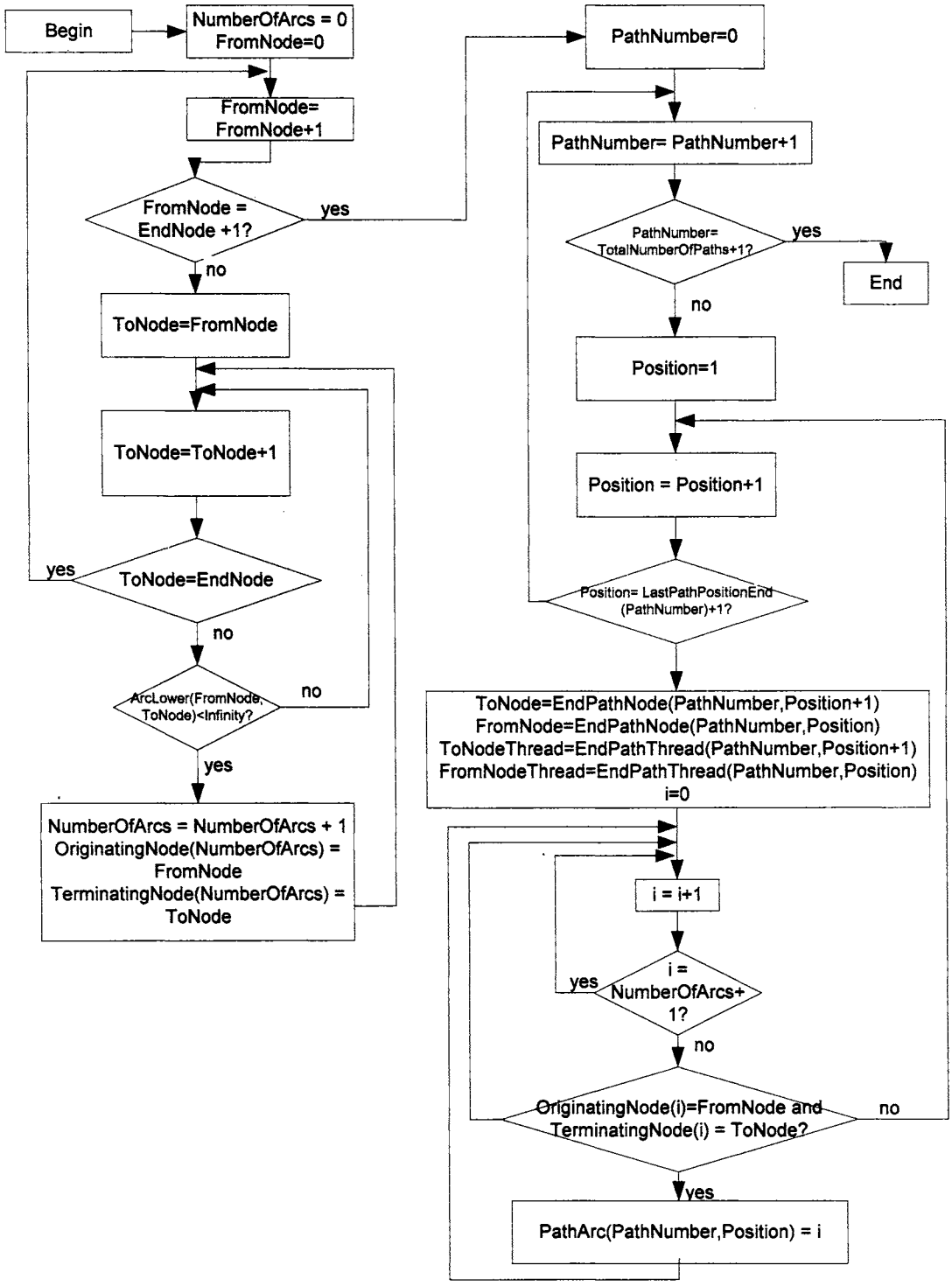
APPENDIX J

FLOW CHART FIND SHARED SUB-PATH INFO



APPENDIX K

FLOW CHART GET ARCS



VITA

Sherri Shearon Avery

Candidate for the Degree of

Doctor of Philosophy

Thesis: QUALITATIVE OPTIMIZATION: DEVELOPMENT OF A METHODOLOGY FOR DETERMINING THE SHORTEST PATH OF A NETWORK WITH INTERVAL-VALUED ARC LENGTHS.

Major Field: Industrial Engineering and Management

Biographical:

Personal Data: Born in Winter Garden, Florida, on September 7, 1965, the daughter of Nathan Eugene Shearon and Jeanne Lee Finch.

Education: Graduation from Enid High School, Enid, Oklahoma in May 1984. Received Bachelor of Arts degree in Mathematics from California State University at Fullerton, Fullerton, California in January 1989. Received Masters of Science degree in Mathematics from Trinity College, Hartford, Connecticut in May 1995. Completed the requirements for the Doctor of Philosophy with a major in Industrial Engineering and Management at Oklahoma State University in December 2003.

Experience: Worked as a computer analyst for Pratt & Whitney Aircraft from 1989 to 1990; taught and tutored in the Math Center at Trinity College from 1992 to 1995; taught as an adjunct lecturer and instructor at the University of Central Oklahoma in the Department of Mathematics and Statistics from 1995 to 2001; worked as a research assistant in the School of Industrial Engineering and Management from 2001 to 2002; taught in the Department of Mathematics at Oklahoma State University in 2002; taught as a graduate teaching assistant in the School of Industrial Engineering and Management in 2003.

Professional Memberships: INFORMS, Institute for Operations Research and the Management Sciences; Alpha Pi Mu, Industrial Engineering Honor Society.