Shortest Path Problems: Multiple Paths in a Stochastic Graph

Melissa Chase
Harvey Mudd College

Shortest Path Problems:

Multiple Paths in Stochastic Graphs

by

Melissa Chase

Ran Libeskind-Hadas, Advisor

Advisor: ____

Second Reader: ____

(Arthur Benjamin)

April 2003

Department of Mathematics

HARVEY MUDD COLLEGE

Abstract

Shortest Path Problems:
Multiple Paths in Stochastic Graphs

by Melissa Chase

April 2003

Shortest path problems arise in a variety of applications ranging from transportation planning to network routing among others. One group of these problems involves finding shortest paths in graphs where the edge weights are defined by probability distributions. While some research has addressed the problem of finding a single shortest path, no research has been done on finding multiple paths in such graphs. This thesis addresses the problem of finding paths for multiple robots through a graph in which the edge weights represent the probability that each edge will fail. The objective is to find paths for n robots that maximize the probability that at least k of them will arrive at the destination. If we make certain restrictions on the edge weights and topology of the graph, this problem can be solved in $O(n \log n)$time. If we restrict only the topology, we can find approximate solutions which are still guaranteed to be better than the single most reliable path.

Table of Contents

List of Figures

Acknowledgments

I would like to thank Professor Ran Libeskind-Hadas for all his help and advice this year. I would also like to thank Professor Arthur Benjamin for encouraging me to pursue the math major and Nathaniel Dirksen for suggesting the revised version of this problem.

Chapter 1

Introduction

I began my research looking at algorithms to find multiple paths through a stochastic graph, a graph in which each edge weight is given by a probability function. Given n robots at some source vertex, the objective is to find paths which will send at least k of those robots to a specified destination vertex, minimizing some objective function. Some possible objective functions might involve minimizing the expected time that the earliest robot would arrive, minimizing the expected average arrival time for the first k robots, or maximizing the probability that at least k robots would arrive by a given time.

Initially, I focused on the problem of finding paths for two robots that minimize the expected arrival time of the first robot to arrive at the destination. I assumed that the weights on the edges would be discrete random variables. On first glance it may appear that the best solution would be to send both robots along the path with the shortest expected travel time. However, in some cases, it is beneficial to divide the robots up, sending one on a slower path that is guaranteed to arrive by a certain time and sending the other on a faster but "riskier" path.

1.1 The Problem is Hard

This problem turned out to be much more complicated than I had anticipated. Most shortest path algorithms in deterministic, static graphs, rely on the fact that any overall shortest path passing through a node will first traverse the shortest path to that node. Unfortunately, this is not true for stochastic graphs.

Furthermore, even computing the expected earliest arrival time given two paths is extremely difficult. Because it represents the expected value of a minimum of two random variables, this requires examining every possible value of each variable and their relative probabilities. When two edges are added together, the possible arrival times at the final vertex are all the possible combinations of one travel time for the first edge and one for the second. Thus, if a path has n edges and each edge has p possible travel times, there may be p^n possible travel times for the entire path. Then finding the earliest arrival time along two such paths requires examining every one of these possible times.

1.2 Revised Problem Statement

I decided instead to look at a simpler but related problem: Instead of having a weight which represents the probability distribution, each edge has only a single probability of failure. The objective is then to plan paths for n robots to maximize the probability that at least k of them reach the designated destination. In this case, it is not best to send all robots on the most reliable path. Sending each robot on a path which is at least slightly different will give some probability of one of them reaching the destination even if one path fails.

Chapter 2

Related Work

There are a variety of different shortest path problems, some of which have been solved, and many of which have not even been considered. I am considering the class of shortest path problems to be those problems which search for a path or several paths optimizing some cost function. Within this class of problems, a graph can have many different types of edge weights, each of which may require a different approach to finding the shortest path. On many types of graphs there are several ways that the total cost of a path can be defined. Furthermore, algorithms can vary with respect to the amount of information which is known when the algorithm is run. Finally, there are several types of algorithms whose goal is not to find a path with the least cost, but which have some other way of defining optimality.

2.1 Graph Weights

The simplest graphs are unweighted, and the shortest path in such a graph merely looks for a path which traverses the fewest edges. Slightly more complicated are those graphs whose edge weights are single values representing the cost of traversing an edge. In this case, the shortest path is the path whose edges have the minimum total cost. Edge weights can be yet more complex and thus require more complex algorithms for finding shortest paths. Some examples of complex edge weights would be those which are time-dependent or stochastic, or even more complicated, a combination of the two.

2.1.1 Time-Dependent Graphs

A separate class of algorithms has been developed to consider the case when the edge weights are not static, but instead change with time. Sometimes this weight can represent the travel time across an edge, but there can also be an additional cost associated with each edge. The weight can change according to some continuous function giving the weight at every time, or a discrete function, where weights are given for a series of time intervals. Another important distinction can be made in the way that algorithms represent the passage of time. Some discretize time into a series of integers, requiring that everything happen exactly on the time increment. Others allow time to be continuous so that events can occur at any time. Note that because the weights change, it may sometimes be advantageous to wait until the weight is lower before traversing an edge.

Time-dependent graphs have been used when it is known how travel times change. Thus, they are useful in transportation applications, to model, for example, freeways which take much longer to traverse during rush-hour than in the middle of the night. They have also been used to model networks of buses and trains where edges can only be traversed at certain points in time. Often, this involves setting the weight of an edge to infinity until a bus is ready to depart. One issue which becomes important in categorizing time-dependent shortest path problems is whether or not a given graph obeys the "Non-Passing Principle" or NPP. A graph with a given set of time-dependent edge weights obeys the NPP if leaving later can never allow one to arrive earlier. If a graph is known to obey this principle, there will never be any reason to wait at any node, because waiting will never allow one to arrive at the destination any earlier. Many algorithms use this fact and require that the given graph obeys the NPP.

2.1.2 *Stochastic Graphs*

One class of algorithms examines paths where the edge weights are not given by a single deterministic value. Instead, with each edge is associated a random variable with a given probability function representing the possible traversal costs of the edge and the likeliness of each cost. This could be a discrete function with a series of costs and their probabilities, or a continuous probability density function. Often, algorithms assume other restrictions on the probabilities, limiting them to be decreasing and linear or exponential, for example. This type of graph is often useful in modeling transportation networks where there is some uncertainty in the amount of traffic that will be on each road, so travel times can only be estimated.

2.1.3 *Stochastic and Time-Dependent Graphs*

Finally, there are graphs whose edge weights are both stochastic and time-dependent. Instead of having one probability distribution or a deterministic weight that varies over different times, these graphs have a probability function that changes over time. This probability function can be discrete or continuous, the way it changes could be discrete or continuous, and time itself could be discrete or continuous.

These graphs can follow a principle analogous to the NPP in deterministic graphs, called "stochastic consistency" [27]. Stochastic consistency means that the probability of arriving by a given time can't be improved by leaving later. As with the NPP, certain algorithms require that the given graphs be stochastically consistent.

2.2 Total Cost

There are a variety of ways for defining the "shortest path." As mentioned above, in a simple non time-dependent, non-stochastic graph, the shortest path is simply the path with the smallest total weight. However, in time-dependent and/or stochastic graphs, the shortest path is often defined differently.

2.2.1 Time-Dependent Graphs

Minimum Time

In most cases, the shortest path is simply the path which takes the least time to traverse. There have been several algorithms proposed for finding the shortest time path in a time-dependent graph, some dealing with discretized, and others with continuous time. Chabini gives an algorithm to find the shortest path when time is discrete and the graph is only time-dependent until some time M. In this case, he expands the graph to a much larger graph with nodes for each time and space combination. Then, he can find the shortest paths from all source nodes at all departure times to the destination in $\theta(SSP + nM + mM)$ time, where n is the number of nodes, m is the number of edges, and SSP is the time required to find a static shortest path [6].

Other algorithms deal instead with the continuous time case. When time is continuous and edge weights are changing, the cost of traversing an edge can be calculated in several different ways. The simplest way is to use the cost at the time of departure from the first node. There are other approaches, however, which take into account any changes in the cost of the edge. Several of these models also guarantee the NPP. Orda and Rom in [19] discuss how one such cost can be converted into the simpler departure time model. Once this conversion is made, if the NPP holds, as it does in this case, a Dijkstra-like algorithm will find the shortest path. Sung, Bell, Seong, and Park, give a similar result using a slightly different version of cost, which runs in $O(n^2 + mv)$ time, where n and m are the number of nodes and edges in the network, and v is the maximum number of weight changes which need to be considered for any one edge [26]. In a similar approach Cooke and Halsey used a modified version of Bellman's algorithm to find the shortest path between any two vertices in a network [7].

Minimum Weight

There are also several algorithms which find the path minimizing some additional time-dependent cost, or a function of that cost and travel time. Thus, such an algorithm might be used if a traveler wanted to spend as little money as possible on a trip, but was not as particular about when he was to arrive. Chabini, in [6], examines the case of this problem where time is discrete, and edge weights are only time-dependent until a given time M. For such a graph, he is able to find optimal paths from every node and at all departure times in $\theta(SSP + nM + mM)$ time, where as before, n is the number of nodes, m is the number of edges, and SSP is the time required to find a static shortest path. He also proves that this time is the best possible for this particular model.

In [1], Ahuja proves that finding the general minimum cost path in a time-dependent network is NP-complete. However, he presents a pseudopolynomial-time algorithm to solve this problem when time is discretized. Orda and Rom pursued this problem further by finding a criteria for which the minimum cost path in the continuous time model is finite, and then describing an algorithm to find that path given a situation where the criteria holds. There is, however, no running time given for this algorithm.

2.2.2 Stochastic Graphs - Utility Functions

There have been a number of works which examine a stochastic graph to find the path with the least expected travel time. For that problem it is possible to replace all the edge weights with their expected values and then solve the problem with a simple deterministic algorithm [16]. However, Loui points out that there is a lot of information lost in this process, and that, often, more than the expected time is important [16]. For example, it may be extremely advantageous to arrive by a specific time, so that taking a path with a longer expected time, but which has a possibility of arriving earlier can be better than taking the path with the least expected time. He first suggests minimizing some function of the upper and lower bounds on the total path weight, however, he says that this strategy is not widely used and points out that it does not work if an edge weight has some probability of being infinite. Instead, he suggests using an idea which he attributes to von Neumann and Morgenstern, which involves using a "utility function" to allow some arrival times to be preferred over others. The value of the utility function at a given time represents how relatively advantageous it would be to arrive at that time. Then the goal of the shortest path algorithm is to maximize the expected value of the utility function.

There are several algorithms which use such a utility function. In [13], Kerr describes an algorithm for maximizing a utility function when the utility function is a "decreasing deadline" function, a function which decreases linearly until some time, then falls to 0. Murthy and Sakar propose a solution to a similar problem in which they use 2 different pruning techniques to reduce the number of paths that must be considered [18]. Eiger, Mirchandani, and Soroush show that whenever the given utility function is linear or exponential, a Dijkstra-like algorithm can be used [8]. Bennett and Bard attempt to find a faster solution given any general utility function by using a heuristic to limit the number of paths considered [2]. Bard and Miller also approach a more complicated version of this problem in which the algorithm is also given a fixed amount of resources to spend on reducing the uncertainty in the graph. Thus, it must first decide where to apply the resources, and then use the resulting information to find the path which maximizes the expected utility. They present a heuristic for solving this problem, but in their implementation, it was not fast enough to be efficient for very large graphs [3].

2.2.3 Stochastic Time-Dependent Graphs

There are several different ways to define the shortest path on a stochastic, time-dependent graph. One of the most common is to find the path with the shortest expected travel time given a specific departure time. Unlike the non-time-dependent case, this cannot necessarily be reduced to a Dijkstra-like algorithm. This is because the weights change, so if one edge changes from very fast to very slow, the possibility of arriving at the first node before the weight change would be better than a guaranteed arrival during the slow period, even if the second expected arrival time was slightly earlier.

The earliest solution to this problem was presented in [10], where Hall gives a high level description of an algorithm for the path with the earliest expected arrival time. Wellman followed up with an optimization of this algorithm which reduces the number of paths considered on a network that obeys the principle of stochastic consistency, and Kelly produced an implementation from which he shows that Wellman's optimization significantly reduces the running time of Hall's algorithm [27, 12]. Kaufman and Smith also proposed an optimization on Hall's algorithm. They use heuristics to find upper and lower bounds on the travel time of the final path, so that many paths needn't be considered [11]. Furthermore, Wellman proposes an approximation algorithm which uses stochastic consistency and stochastic dominance to find approximate shortest paths which are contained within continuously tightening upper and lower bounds [15]. There is no running time given for this algorithm, but it is guaranteed to approach the optimal solution as it is given more time. These are all continuous time algorithms. However, when time is discretized, algorithms can run much quicker. Thus, Pretolani's algorithm can find the best route for such a problem in $O(k)$ time, where k is the total number of arrival times along all of the edges [24]. Unfortunately, this algorithm is based around the use of hypergraphs, and is therefore not space efficient.

However, not all shortest path algorithms for time-dependent stochastic graphs aim to minimize the expected arrival time. As we have observed above, this is not always the desired objective. Thus, Miller-Hooks and Mahmassani designed an algorithm focused on finding the "least possible time path", or the path with the possibility of taking the least time [17]. As with the "adaptive" algorithms described below, this algorithm gives a path from each node for each departure time. In this case, it gives both the least possible time path and the probability of achieving this travel time, thus providing an alternative to the least expected time path.

2.3 Amount of Information Given

In a static or time-dependent deterministic problem, all of the graph parameters are already known when the algorithm is run. Thus, an algorithm is able to return one path which is guaranteed to be as short as possible. However, there are other algorithms that run based on the fact that all the parameters are not given beforehand.

2.3.1 Adaptive Plan Algorithms

In stochastic graphs, while the probabilities for traveling across a certain edge are known, the exact time the traversal will take is not. Thus, several of the algorithms dealing with stochastic graphs have been designed to return not a single path, but a strategy for traversing the graph based on the arrival time at each node. Following such a strategy has been shown to give traversal times at least as short as any single shortest path [10].

Boyan and Mitzenmacher [4] have developed an algorithm for a bus network, where in a graph of bus lines and bus stops, at each stop each bus has a specific continuous probability function describing when it is likely to arrive. It requires only that these probability functions have an increasing failure rate, meaning that as time goes on, if an edge still is not traversable, the probability the it will soon be traversable is always increasing, i.e. $P(x \geq k + t | x \geq t) \leq P(x \geq k)$ where x, t, and k, are all times and $P(t)$ is the probability of being able to traverse the edge at time t. Given this information, the algorithm returns an optimal travel plan of the form "take bus 1 whenever it arrives; take bus 2 if it arrives before time 2; take bus 3 if it arrives before time 3; and so on." This algorithm has complexity $O(hSQ \log(Q))$ where h is the maximum allowable number of bus changes, S is the number of stations, and Q is the maximum number of buses per station.

The plan algorithms are not restricted to static graphs. Hall, in [10], gives a very high level description of an algorithm which runs on stochastic, time-dependent graphs and returns the best edge to traverse for each range of arrival times at each node. Later, an optimization and more concrete description of this algorithm was developed by Wellman, Larson,and Ford [27]. They require stochastic consistency and give an optimal solution on a network of buses in $O(B^2Dn)$ time and $O(BDn)$ space, where B is the maximum number of buses per edge, D is the maximum out-degree of a node, and n is the number of nodes in the graph.

Similar adaptive plan algorithms have been developed for minimizing a utility function over stochastic graphs. In [13], Kerr's algorithm returns a optimal traversal plan which maximizes the value of a decreasing deadline utility function in a static stochastic graph. Once again, this plan gives the best edge to traverse given the time of arrival at a specific node.

2.3.2 Partially Observed Graphs

Other algorithms consider graphs in which even less information is known. In particular, a family of algorithms considers the case when the form of the graph itself is not entirely known.

Pemberton and Korf examine the case where the only thing known is a rough estimate of how far a given node is from the destination. Nodes that have already been explored and those adjacent to them are the only ones that are visible. The algorithm explores the graph, discovering the positions of nodes and the costs of the adjacent edges. It traces a path which, while not necessarily optimal, is guaranteed to eventually reach the destination in a worst case of $O(n^2)$ time and $O(n+m)$ space in a graph with n nodes and m edges [22].

2.4 Algorithm Goals

Finally, not all of the algorithms have a simple, or an absolute shortest path as their goal. Sometimes it is useful to find several good paths, or to have algorithms which, once they have found one shortest path, take very little time to find the solution to a slightly different problem.

2.4.1 All Criterion Paths

Often it is useful to have, not just one overall shortest path, but a set of relatively short paths from which to choose. One set of alternatives is the set of *k-shortest* paths. For a given number k, these are the k shortest paths which differ from the optimal shortest path by at least one edge. Brander and Sinclair in [5] describe 4 different algorithms for finding these k paths in static deterministic graphs. They describe algorithms by Yen and Lawler which take $O(kn^3)$ time, and faster algorithms by Katoh and Hoffman which take $O(kn^2)$ time. Scott goes a bit farther and points out that often, paths that differ by only one edge are not different enough. Instead, she proposes finding *k-similar* paths: paths with at most k edges in common with the original path. She mentions that this problem is NP-hard, but explains how to use Lagrangian Relaxation to approximate the solution [25].

2.4.2 Reoptimization

It is also often useful to find the shortest paths in several graphs which are only slightly different. Nguyen, Pallotino, and Scutella describe an algorithm which, given a graph and the shortest path from the source to the destination, finds the shortest path to the same destination from a different source. They also include an approximation optimization of this algorithm. This gives a complexity of $O(n^2 + km)$, where n is the number of nodes, m is the number of edges, and k is the number of shortest paths that have already been calculated [21].

Chapter 3

Two Path Problem

This thesis considers the following problem: We are given a directed graph in which the weight of each edge represents the reliability, i. e. the probability that the edge will not fail. If two paths traverse the same edge, they will either both fail or both successfully complete the traversal. Given a source and destination on that graph, we would like to find two paths from the source which maximize the probability that at least one will reach the destination. For the most of this discussion, we will consider only series-parallel graphs.

A directed graph G is *two-terminal series parallel*, with terminals s and t, if it can be produced by a sequence of the following operations:

1. Create a new graph, consisting of a single edge directed from s to t.

2. Given two two-terminal series parallel graphs X and Y, with terminals $s_X, t_X, s_Y,$ and t_Y, form a new graph $G = P(X, Y)$ by identifying $s = s_X = s_Y$ and $t = t_X = t_Y$. This is known as the *parallel composition* of X and Y.

3. Given two two-terminal series parallel graphs X and Y, with terminals s_X, t_X, s_Y, and t_Y, form a new graph $G = S(X, Y)$ by identifying $s = s_X, t_X = s_Y,$ and $t = t_Y$. This is known as the *series composition* of X and Y. [9]

Lemma 3.0.1 *On a series parallel graph, one of the two paths in the optimal solution will be one of the single most reliable paths.*

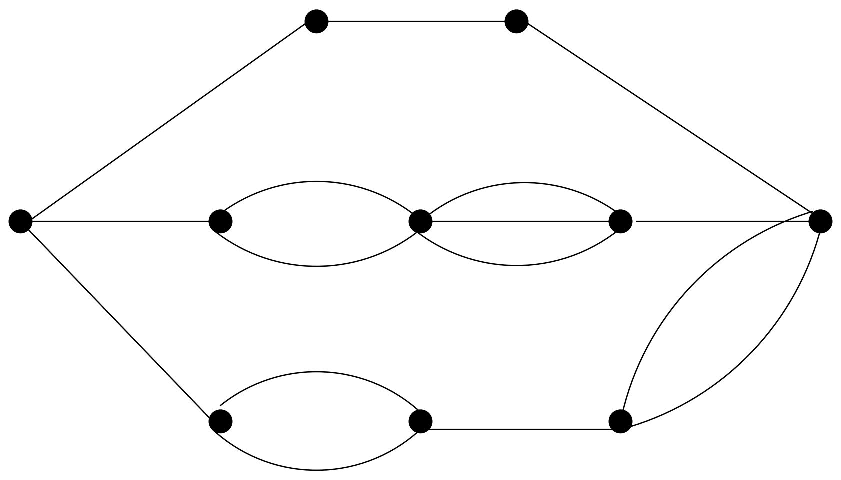

Figure 3.1: A series parallel graph

Proof: Let C be the most reliable path from the source, v_0 to the destination, v_n. Let A and B be an optimal pair of paths such that neither A nor B is as reliable as C. Let the subpath of A from node s to node t be the shortest subpath such that s and t both appear in C and A is less reliable than C from s to t. Note that the two subpaths will share no edges: otherwise we could remove those edges leaving a shorter segment over which C was more reliable than A. Thus, these two subpaths are arranged in parallel.

If B does not intersect C between s and t:

Let A′ be the path which follows A from v_1 to s, follows C from s to t, and then follows A again from t to v_n. Originally, the probability of either path A or path B arriving at v_n was $P(A \vee B) = P(A) + P(B) - P(AB)$. Now $P(A' \vee B) = P(A') + P(B) - P(A'B)$.

Let $P(A_{rest})$ be the probability that path A does not fail between v_1 and s or between t and v_n. Let $P(BA_{rest})$ be the probability that B doesn't fail anywhere and that A does not fail between v_1 and s and between t and v_n. Let $P(A_{st})$ and

$P(C_{st})$ be the probabilities that A and C do not fail between s and t.

$$\begin{aligned}
P(A' \vee B) &= P(A') + P(B) - P(A'B) \\
&= P(A_{rest})P(C_{st}) + P(B) - P(C_{st})P(BA_{rest}) \\
&\geq P(A_{rest})P(A_{st}) + P(B) - P(A_{st})P(BA_{rest}) \\
&\geq P(A) + P(B) - P(AB) \\
&= P(A \vee B)
\end{aligned}$$

Thus, A' and B are a more reliable pair of paths than A and B, contradicting our initial assumption.

If B intersects C between s and t:

B cannot intersect A because these two paths are parallel in that interval. Similarly, B must contain nodes s and t.

Let B′ be the path which follows B from v_1 to s, C from s to t, and B from t to v_n.

Let $P(B_{rest})$ be the probability that B does not fail between v_1 and s or between t and v_n, $P(B_{st})$ be the probability that B does not fail between s and t, and $P(AB_{rest})$ be the probability that A does not fail anywhere and that B does not fail between v_1 and s or between t and v_n.

$$\begin{aligned}
P(A \vee B') &= P(A) + P(B') - P(AB') \\
&= P(A) + P(B_{rest})P(C_{st}) - P(C_{st})P(AB_{rest}) \\
&\geq P(A) + P(B_{rest})P(B_{st}) - P(B_{st})P(AB_{rest}) \\
&\geq P(A) + P(B) - P(AB) \\
&= P(A \vee B)
\end{aligned}$$

Thus, A and B' are a more reliable pair of paths than A and B, contradicting our initial assumption. $\square$

Chapter 4

Chain topology

We will begin by considering this problem on very simple graphs. We will examine graphs consisting of a series of nodes $v_1 \ldots v_n$, where v_1 is the source and v_n is the destination, and in which each pair of consecutive nodes can be connected by any number of edges. We will refer to this as a *chain topology*. In this case there are several observations which can be made.

4.1 Basic Observations

Observation 4.1.1 *One path will contain the most reliable edge between each pair of nodes.*

This follows directly from Lemma 3.0.1.

Observation 4.1.2 *The second path will always take one of the two most reliable edges. The rest of the edges will never be used.*

Proof: Let the edges between v_i and v_{i+1} have probabilities of not failing $e_1 \ldots e_k$, where $e_j > e_{j+1}$ for all j, $1 \leq j \leq n-1$. e_1 will be used by at least 1 path by

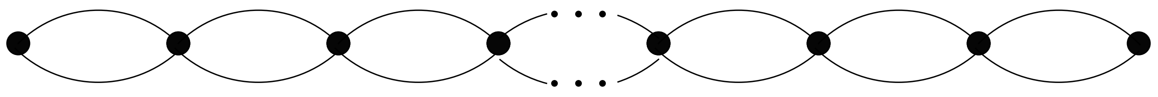

Figure 4.1: A chain graph, in this case with exactly two edges between each pair of adjacent nodes.

Lemma 3.0.1. Suppose that in the optimal pair of paths, the second path uses some e_j, $j > 2$. Let P_1 and P_2 be the probability that the first and second paths arrive at v_n, and P_{12} be the probability that both arrive. Thus, the probability that at least one of the paths does not fail will be $P_1 + P_2 - P_{12}$. Further, obviously P_2 is greater than P_{12}. The probability that a path does not fail is the product of the reliability of all of its edges. If we replace e_j with e_2, the probability that the second path arrives will be $P_2' = P_2 * \frac{e_2}{e_j}$ and the probability that both paths arrive will be $P_{12}' = P_{12} * \frac{e_2}{e_j}$. The probability that at least one path arrives at v_n will be $P_1 + P_2' + P_{12}' = P_1 + \frac{e_2}{e_j}(P_2 - P_{12}) > P_1 + P_2 - P_{12}$, since $\frac{e_2}{e_j} > 1$. Thus, it is more likely that at least one path will arrive at v_n if the second path takes e_2 instead of e_j, which contradicts the original assumption that one of the optimal paths would take e_j. $\square$

This means that we can simplify any such graph by removing all but the two most reliable edges between any pair of nodes.

Observation 4.1.3 *The order of the node pairs is insignificant.*

This follows from the fact that multiplication is associative, and the probability of several paths not failing is just the product of the probabilities that each of the edges won't fail.

Given this framework, let a_i and b_i be the most reliable edges between v_i and v_{i+1}, and let A_i and B_i be their respective weights such that $A_i \geq B_i$. Further, if $A_i = 1$ or $B_i = 0$, any optimal pair of paths will remain optimal if both paths traverse a_i. In both cases, the optimal path in the rest of the graph is the same as that in the node chain where nodes v_i and v_{i+1} have been combined. Thus, once we have identified all such node pairs, we have only to find the optimal path in the rest of the graph. From here on we will assume that $0 < B_i \leq A_i < 1$.

Now, we know that at least one path will take $a_1, \dots a_{n-1}$ by Observation 4.1.1 . Call two paths p_1 and p_2. The probability of p_1 arriving at v_n is just the product of all the reliabilities of the edges in each path. The same is true for p_2. The probability that both paths arrive is the product of the probability that both paths will successfully traverse each pair of nodes. Between v_i and v_{i+1}, if both paths take a_i, this probability will be A_i, and if the second path takes b_i, this probability will be A_iB_i. Now, the probability that at least one path arrives at the destination is:
$P_1 + P_2 - P_{12} = \prod_{i<n} A_i + \prod_{a_i \in p_2} A_i \prod_{b_i \in p_2} B_i + \prod_{i<n} A_i \prod_{b_i \in p_2} B_i$

Lemma 4.1.4 *Edge $b_i \in p_2$ for some optimal pair of paths p_1 and p_2 iff $\frac{B_i}{A_i}\frac{1-A_ip}{1-p} \geq 1$ where $p = \prod_{b_j \in p_2, j \neq i} A_j$*

Proof: If we examine just one pair of edges, a_i and b_i, let P'_1 be the probability that the first path does not fail through the rest of the graph, P'_2 be the probability that the second path does not fail through the rest of the graph, and P'_{12} be the probability that neither path fails over the rest of the edges. Then, if the second path includes b_i, the probability that at least one path will reach the destination is $A_iP'_1 + B_iP'_2 - A_iB_iP'_{12}$. If the second path includes a_i, the probability is $A_iP'_1 + A_iP'_2 - A_iP'_{12}$. Thus, the path taking b_i will be better than the path taking a_i iff

$$\begin{aligned} A_iP'_1 + B_iP'_2 - A_iB_iP'_{12} &> A_iP'_1 + A_iP'_2 - A_iP'_{12} \\ B_iP'_2 - A_iB_iP'_{12} &> A_iP'_2 - A_iP'_{12} \\ B_iP'_2(1 - A_i\frac{P'_{12}}{P'_2}) &> A_iP'_2(1 - \frac{P'_{12}}{P'_2}) \end{aligned}$$

Let $p = \frac{P'_{12}}{P'_2}$

$$\begin{aligned} B_iP'_2(1 - A_ip) &> A_iP'_2(1 - p) \\ \frac{B_i}{A_i}\frac{1 - A_ip}{1 - p} &> 1 \end{aligned}$$

$$
\begin{aligned}
p &= \frac{P'_{12}}{P'_2} \\
p &= \frac{\prod_{j \neq i} A_j \prod_{b_j \in p_2, j \neq i} B_i}{\prod_{a_j \in p_2, j \neq i} A_j \prod_{b_j \in p_2 j \neq i} B_j} \\
p &= \prod_{j \neq i, a_j \notin p_2} A_j
\end{aligned}
$$

□

4.2 Maximum Number of Split Edge Pairs

Let a *split edge pair* in a pair of paths denote a pair of edges a_i and b_i such that the first path takes a_i and the second path takes b_i. Now, let $M = max_{1 \leq i \leq n} A_i$ and $m = \max_{1 \leq i \leq n} \frac{B_i}{A_i}$

Lemma 4.2.1 *The maximum possible number of split edge pairs in an optimal pair of paths is* $\lceil \log_M (1-m) \rceil$

Proof: Let x be the number of splits in an optimal pair of paths. Let a_i, b_i be one of the edge pairs which will be split. That means $\frac{b_i}{a_i} \frac{1-A_i p}{1-p} > 1$. Since there are x other split edges, and since $p = \prod_{b_j \in p_2, j \neq i} A_j$, we know $p \leq M^{x-1}$. Note that the quantity $\frac{1-A_i p}{1-p}$ increases as p approaches 1.
That means that
$1 < \frac{1-A_i p}{1-p} \leq \frac{1-A_i M^{x-1}}{1-M^{x-1}} < \frac{1}{1-M^{x-1}}$
Also, $\frac{b_i}{a_i} \leq m$ by definition. Thus, since b_i, a_i is a split edge pair,

$$
\begin{aligned}
1 \leq \frac{B_i}{A_i} \frac{1-A_i p}{1-p} &< m \frac{1}{1-M^{x-1}} \\
1 &< m \frac{1}{1-M^{x-1}} \\
m &> 1 - M^{x-1} \\
1 - m &< M^{x-1}
\end{aligned}
$$

Since $1-m, M^{x-1} < 1$,

$$\begin{aligned} \log_M(1-m) &> x-1 \\ \log_M(1-m)+1 &> x \\ \lceil \log_M(1-m) \rceil &\geq x \end{aligned}$$

Thus, the number of split edges can be at most $\lceil \log_M(1-m) \rceil$. □

Similarly, if we define $f = \min_{1\leq i\leq n} A_i$ and $g = \min_{1\leq i\leq n} \frac{B_i}{A_i}$, we can show that the minimum number of split edges is $\lfloor \log_f(1-g) \rfloor$.

4.3 Sorting Edge Pairs

Recall that in an optimal pair of paths, the a_i, b_i edge pair will be a split edge pair (i.e. the first path will take a_i and the second path will take b_i) iff $\frac{B_i}{A_i}\frac{1-A_ip}{1-p} \geq 1$ where $p = \prod_{b_j \in p_2, j\neq i} A_j$. Let x be the number of split edge pairs. Then $p < M^x$.

Lemma 4.3.1 *If $A_j > A_i$, edge pair a_j, b_j is split in an optimal pair of paths, and $\frac{B_i}{A_i} > \frac{B_j}{A_j}$,then a_i, b_i is also split.*

Proof: If $A_j > A_i$, then $\frac{1-A_ip}{1-A_jp} > 1$.

$$\begin{aligned} \frac{B_i}{A_i}\frac{1-A_ip}{1-p} &= \frac{\frac{B_i}{A_i}}{\frac{B_j}{A_j}}\frac{1-A_ip}{1-A_jp} * \frac{B_j}{A_j}\frac{1-A_jp}{1-p} \\ &\geq \frac{\frac{B_i}{A_i}}{\frac{B_j}{A_j}}\frac{1-A_ip}{1-A_jp} * 1 \text{ since } a_j, b_j \text{ is split.} \\ &> 1 \end{aligned}$$

Thus, a_i, b_i is split by Lemma 4.1.4. □

Lemma 4.3.2 *If $A_j > A_i$, edge pair a_j, b_j is split in an optimal pair of paths, and $\frac{B_j}{A_j} \leq (1-M^x)\frac{B_i}{A_i}$, then a_i, b_i is also split.*

Proof: Suppose this is not true. That means, a_j, b_j is split, but a_i, b_i is not. When x-1 other edges have been split, $p \leq M^{x-1}$.
Thus, $\frac{1-A_i p}{1-A_j p} \leq \frac{1-A_i M^{x-1}}{1-A_j M^{x-1}} < \frac{1}{1-A_j M^{x-1}} \leq \frac{1}{1-M^x}$.

$$\begin{aligned}
\frac{B_j}{A_j}\frac{1-A_j p}{1-p} &= \frac{\frac{B_j}{A_j}}{\frac{B_i}{A_i}}\frac{1-A_j p}{1-A_i p} * \frac{B_i}{A_i}\frac{1-A_i p}{1-p} \\
&\leq \frac{\frac{B_j}{A_j}}{\frac{B_i}{A_i}}\frac{1-A_j p}{1-A_i p} * 1 \text{ since } a_i, b_i \text{ is not split.} \\
&< \frac{\frac{B_j}{A_j}}{\frac{B_i}{A_i}}\frac{1}{1-M^x} \\
&\leq (1-M^x)\frac{1}{1-M^x} \\
&= 1
\end{aligned}$$

Thus, a_j, b_j is not split by Lemma 4.1.4, in contradiction to our original statement.□

Let k be the maximum value of $\frac{\frac{B_i}{A_i}}{\frac{B_j}{A_j}} < 1$ over all i and j.

Lemma 4.3.3 *If $k \leq 1 - M^x$ and $\frac{B_i}{A_i} > \frac{B_j}{A_j}$ and a_j, b_j is split then a_i, b_i is split.*

Proof: If $A_j > A_i$, then this is a restatement of Lemma 4.3.1
If $A_j < A_i$:
Since $\frac{B_i}{A_i} > \frac{B_j}{A_j}$,
$\frac{B_j}{A_j} \leq k\frac{B_i}{A_i} \leq (1-M^x)B_i A_i$
Thus, by Lemma 4.3.2, a_i, b_i is split.

Thus, in chain graphs in which $k \leq 1 - M^x$, we can find an optimal pair of paths using the following algorithm:
Sorting Algorithm:

1. Sort the edge pairs by $\frac{B_i}{A_i}$.

2. Split the first edge pair.

3. Compute $\frac{B_i}{A_i}\frac{1-A_i p}{1-p}$ in which $p = \prod_{1 \le h \le i} A_i$ for each successive edge pair a_i, b_i.

4. If a_l, b_l is the first edge pair for which this quantity is less than 1, then an optimal second path will take b_i for all $i, 1 \le i < l$ and a_i for all $i, l \le i < n$.

Note, there is always at least 1 edge split so if $k < 1 - M$, this algorithm is always successful.

This algorithm will take $O(n \log n)$ to sort the list and $O(n)$ to compute $\frac{B_i}{A_i}\frac{1-A_i p}{1-p}$ for each edge pair. Thus, the total running time is $O(n \log n)$.

4.4 Semi Ordered Edge Pairs

If $k^w < 1 - M^x$ for very small w and if all $\frac{B_i}{A_i}$ are distinct, a similar approach can be used. First we sort the edge pairs by $\frac{B_i}{A_i}$. Then we find two paths using the sorting algorithm given above. If we say that each ratio $\frac{B_i}{A_i}$ is distinct, if $\frac{B_i}{A_i} < \frac{B_j}{A_j}$, there are at most $w - 1$ edge pairs, a_j, b_j, such that $(1 - M^x)\frac{B_j}{A_j} < \frac{B_i}{A_i}$. Thus, there are at most $w - 1$ edge pairs a_j, b_j which will appear before a_i, b_i in the sorted edge pair list, for which the optimal pair of paths might split a_i, b_i without splitting a_j, b_j. If the sorting algorithm splits the first x edge pairs, we know we have only to examine paths which do not split some subset of the last w of the split edges. There will be 2^w subsets, but if w is a small constant, this does not increase the running time significantly.

4.5 Approximations

If $k > 1 - M^c$, and all of the ratios $\frac{B_i}{A_i}$ are distinct, we can use this algorithm to get an approximate solution.

Let a *skipped edge pair* be an edge pair a_i, b_i which is not split, but where there is some $j > i$ in the sorted edge list such that a_j, b_j is split. In the optimal path, let s be the number of skipped edge pairs.

Lemma 4.5.1 *The Sorting Algorithm gives a pair of paths which is at least* $\frac{1+m^c(1-M^c)}{1+m^c k^s}$ *optimal*

Proof: Suppose OPT's solution has c split edges and s skipped edges. Let ALG_1 and ALG_2 be the pair of paths generated by the Sorting Algorithm. This is the best pair of paths with no skipped edges. Consider the pair of paths P_1, P_2 which have c split edges and no skipped edges. This pair must be less reliable than ALG_1 and ALG_2. Since OPT has c split edges an s skipped edges, the last split edge in OPT will be edge pair a_{c+s}, b_{c+s}. If a_i, b_i appears after a_j, b_j in the sorted list, then we know $\frac{B_i}{A_i} < \frac{B_j}{A_j}$. From our definition of k, this means that $\frac{B_i}{A_i} \leq k\frac{B_j}{A_j}$. Thus, $\frac{B_{c+s}}{A_{c+s}} \leq k^s \frac{B_c}{A_c}$. This also means that the ith split edge, a_y,b_y, in OPT's paths will have $\frac{B_y}{A_y} \leq \frac{B_i}{A_i}$ which will be the ith split edge in ALG_1 and ALG_2.

The ratio of $ALG_1$and ALG_2's reliability to that of OPT will be:

$$\begin{aligned}
\frac{P(ALG_1 \vee ALG_2)}{P(OPT_1 \vee OPT_2)} &= \frac{P(ALG_1) + P(ALG_2) - P(ALG_1 ALG_2)}{P(OPT_1) + P(OPT_2) - P(OPT_1 OPT_2)} \\
&= \frac{\prod_{1\le h\le n-1} A_h + \prod_{1\le h\le c} B_h \prod_{c<h\le n-1} A_h - \prod_{1\le h\le n-1} A_h \prod_{a\le h\le c} B_h}{\prod_{1\le h\le n-1} A_h + \prod_{b_h\in OPT_2} B_h \prod_{a_h\in OPT_2} A_h - \prod_{1\le h\le n-1} A_h \prod_{b_h\in OPT_2} B_h} \\
&= \frac{1 + \prod_{1\le h\le c} \frac{B_h}{A_h} - \prod_{a\le h\le c} B_h}{1 + \prod_{b_h\in OPT_2} \frac{B_h}{A_h} - \prod_{b_h\in OPT_2} B_h} \\
&\ge \frac{1 + \prod_{1\le h\le c} \frac{B_h}{A_h} - \prod_{a\le h\le c} B_h}{1 + \prod_{b_h\in OPT_2} \frac{B_h}{A_h}} \\
&= \frac{1 + \prod_{1\le h\le c} \frac{B_h}{A_h}(1 - \prod_{a\le h\le c} A_h)}{1 + \prod_{b_h\in OPT_2} \frac{B_h}{A_h}} \\
&\ge \frac{1 + \frac{B_c}{A_c} \prod_{1\le h\le c-1} \frac{B_h}{A_h}(1 - M^c)}{1 + \frac{B_{c+s}}{A_{c+s}} \prod_{b_h\in OPT_2, h\ne c+s} \frac{B_h}{A_h}} \\
&\ge \frac{1 + \frac{B_c}{A_c} \prod_{1\le h\le c-1} \frac{B_h}{A_h}(1 - M^c)}{1 + k^s \frac{B_c}{A_c} \prod_{b_h\in OPT_2, h\ne c+s} \frac{B_h}{A_h}} \\
&\ge \frac{1 + m \prod_{1\le h\le c-1} \frac{B_h}{A_h}(1 - M^c)}{1 + k^s m \prod_{b_h\in OPT_2, h\ne c+s} \frac{B_h}{A_h}} \\
&\ge \frac{1 + m \prod_{1\le h\le c-1} \frac{B_h}{A_h}(1 - M^c)}{1 + k^s m \prod_{1\ne h\ne c-1} \frac{B_h}{A_h}} \\
&\ge \frac{1 + m^c(1 - M^c)}{1 + k^s m^c}
\end{aligned}$$

□

Note: There must be at least one skipped edge pair, so this algorithm is at least $\frac{1+m^c(1-M^c)}{1+m^c k}$ optimal. Further, at least one edge must be split, so even if c is unknown, this algorithm is at least $\frac{1+m(1-M)}{1+mk}$ as reliable as an optimal solution.

Recall that we defined w to be the largest number such that $k^w \le 1 - M^c$. Thus, this ratio is at least $\frac{1+m^c k^w}{1+m^c k^s}$. If w is small, this is a fairly close bound.

We can compute the best pair of paths with at most q skipped edge pairs in $O(n^q)$ time by skipping each possible subset of q or less edges and computing the best paths(using the Sorting Algorithm). Further, if all of the ratios ($\frac{B_i}{A_i}$) are distinct, this can be done in $O(nw^{q-1})$ time. This is because, as discussed in the previous section, we need only consider subsets which skip the last w of the edges identified by the algorithm.

If we have found the best pair of paths with at most q skipped edge pairs, then either we have found the optimal paths, or the optimal paths have at least $q+1$ skipped edges. Then, we have a pair of paths which is at least $\frac{1+mk^w}{1+mk^{q+1}}$ optimal.

Thus, in $O(n^q)$ time we can compute a pair of paths which is at least $\frac{1+mk^w}{1+mk^{q+1}}$ as reliable as optimal.

Chapter 5

More Complicated Topologies

We have examined this problem on simple chain graphs. Now we examine the problem on more complicated graphs.

5.1 Additional Uncrossed Edges

Consider a chain graph in which edges have been added connecting some nonadjacent vertices with the restriction that these edges cannot cross: Let e_1 and e_2 be two additional edges between vertices in the chain connecting v_a to v_b and v_c to v_d, where $a < b$ and $c < d$. These edges would be considered crossed if $a < c < b < d$ or $c < a < d < b$. Requiring the edges to be uncrossed means that the graph will still be series parallel.

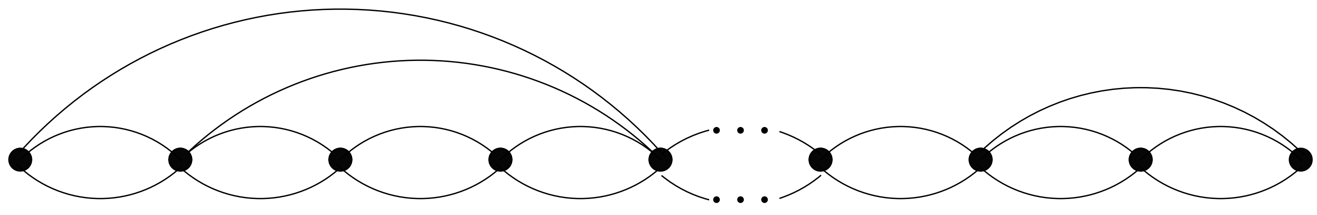

Figure 5.1: An addition to the chain graph: in this case we have added a set of edges connecting nodes along the chain, while still requiring that the graph be series-parallel

Let c be an edge with weight C which spans the same set of nodes as $\{a_s, \ldots, a_t\}$.

Lemma 5.1.1 *If $C \leq \prod_{s \leq i \leq t} B_i$, c will never be used.*

Proof: Let P_1, P_2 be an optimal pair of paths which includes c where $C \leq \prod_{s \leq i \leq t} B_i$. P_1 will take $a_s \ldots a_t$ since that is part of the most reliable path. Thus, P_2 takes c. However, consider the result of replacing c with $b_s \ldots b_t$ in P_2. This increases the reliability of P_2 without increasing the degree to which P_1 and P_2 are interdependent. Thus, we can increase the reliability of the pair P_1, P_2 by replacing c with $b_s \ldots b_t$, which means these paths are not optimal. $\square$

Lemma 5.1.2 *If $C \geq \prod_{s \leq i \leq t} A_i$ then c will always be used.*

This follows by Lemma 3.0.1 directly from the fact that c is then part of the single optimal path. This also simplifies the graph, as we can replace c, $a_s \ldots, a_t$, and $b_s \ldots b_t$ with a single edge pair of weights C and $\prod_{s \leq i \leq t} A_i$.

Since those cases are relatively simple, let's assume that $\prod_{s \leq i \leq t} B_i < C < \prod_{s \leq i \leq t} A_i$. Let x be a minimum for the number of edges a_i which do not appear in the second path.

Lemma 5.1.3 *The second path will use c instead of $b_s \ldots b_{s+r}, a_{s+r+1}, \ldots a_t$ iff* $\frac{C}{\prod_{s \leq i \leq s+r} B_i \prod_{s+r+1 \leq i \leq t} A_i} \frac{1-p\prod_{s \leq i \leq t} A_i}{1-p\prod_{s \leq i \leq s+r} A_i} \geq 1$ *where* $p = \prod_{b_j \in p_2, j \neq s, \ldots s+r} A_j$

This follows in exactly the same form as the proof in chapter 4.

Lemma 5.1.4 *if $a_s, \ldots a_{s+r}$ are split and $a_{s+r+1} \ldots a_t$ are not and if* $\frac{C}{A_x \ldots A_t} \geq \frac{1}{1-M^{x-r}} \prod_{s \leq i \leq s+r} (\frac{B_i}{A_i})^2$*, then the second path will take edge c instead of $b_s \ldots b_{s+r}$.*

Proof: In this case:

$$\begin{aligned}
& \frac{C}{\prod_{s\leq i\leq t} A_i} \prod_{s\leq i\leq r+s} (\frac{A_i}{B_i}) \frac{1-p\prod_{s\leq i\leq t} A_i}{1-p\prod_{s\leq i\leq s+r} A_i} \\
\geq \; & \frac{1}{1-M^{x-r}} \prod_{s\leq i\leq s+r} (\frac{B_i}{A_i})^2 \prod_{s\leq i\leq s+r} \frac{A_i}{B_i} \frac{1-p\prod_{s\leq i\leq t} A_i}{1-p\prod_{s\leq i\leq s+r} A_i} \\
= \; & \frac{1}{1-M^{x-r}} \prod_{s\leq i\leq s+r} \frac{B_i}{A_i} \frac{1-p\prod_{s\leq i\leq t} A_i}{1-p\prod_{s\leq i\leq s+r} A_i} \\
= \; & \frac{1}{1-M^{x-r}} \prod_{s\leq i\leq s+r} \frac{B_i}{A_i} \frac{1-p\prod_{s\leq i\leq s+r} A_i}{1-p} \frac{(1-p\prod_{s\leq i\leq t} A_i)(1-p)}{(1-p\prod_{s\leq i\leq s+r} A_i)^2} \\
\geq \; & \frac{1}{1-M^{x-r}} \prod_{s\leq i\leq s+r} \frac{(1-p\prod_{s\leq i\leq t} A_i)(1-p)}{(1-p\prod_{s\leq i\leq s+r} A_i)^2} \\
\geq \; & \frac{1}{1-M^{x-r}} \prod_{s\leq i\leq s+r} \frac{1-p}{1-p\prod_{s\leq i\leq s+r} A_i} \\
\geq \; & \frac{1}{1-M^{x-r}} (1-M^{x-r}) \\
= \; & 1.
\end{aligned}$$

Thus, the second path will include c. □

Lemma 5.1.5 *If $a_s, \ldots a_{s+r}$ are split and $a_{s+r+1} \ldots a_t$ are not and if $\frac{C}{\prod_{s\leq i\leq t} A_i} \leq \frac{1}{1-M^x} \prod_{s\leq i\leq s+r} \frac{B_i}{A_i}$, then the second path will still take $b_s \ldots b_{s+r}$.*

Proof: In this case:

$$\begin{aligned}
\frac{C}{\prod_{s\le i\le t} A_i}\prod_{s\le i\le r+s}(\frac{A_i}{B_i})\frac{1-p\prod_{s\le i\le t}A_i}{1-p\prod_{s\le i\le s+r}A_i} &\le \frac{1}{1-M^x}\prod_{s\le i\le s+r}(\frac{B_i}{A_i})\prod_{s\le i\le s+r}\frac{A_i}{B_i}\frac{1-p\prod_{s\le i\le t}A_i}{1-p\prod_{s\le i\le s+r}A_i}\\
&= \frac{1}{1-M^x}\frac{1-p\prod_{s\le i\le t}A_i}{1-p\prod_{s\le i\le s+r}A_i}\\
&\le \frac{1}{1-M^x}\frac{1-M^{x-r}\prod_{s\le i\le t}A_i}{1-M^{x-r}\prod_{s\le i\le s+r}A_i}\\
&\le \frac{1}{1-M^x}\frac{1-M^{x-r}M^r\prod_{s+r+1\le i\le t}A_i}{1-M^{x-r}M^r}\\
&\le \frac{1}{1-M^x}\frac{1}{1-M^x}\\
&= 1
\end{aligned}$$

Thus, the second path will not include c. □

Lets extend the definition of k to be the maximum ratio less than one between two parallel paths between the same pair of nodes.

Lemma 5.1.6 *If $k \le 1-M^{x-r}$ then, if $\frac{C}{\prod_{s\le i\le t}A_i} < \prod_{s\le i\le s+r}\frac{B_i}{A_i}$ the second path will take $b_s \ldots b_{s+r}$ otherwise it will take c.*

Proof: If $\frac{C}{\prod_{s\le i\le t}A_i} < \prod_{s\le i\le s+r}\frac{B_i}{A_i}$

Then,

$$\begin{aligned}
\frac{C}{\prod_{s\le i\le t}A_i} &\le k\prod_{s\le i\le s+r}\frac{B_i}{A_i}\\
&\le (1-M^{x-r})\prod_{s\le i\le s+r}\frac{B_i}{A_i}\\
&\le (1-M^{x})\prod_{s\le i\le s+r}\frac{B_i}{A_i}
\end{aligned}$$

Thus, by Lemma 5.1.5, the second path will take $b_s \ldots b_{s+r}$.
If $\frac{C}{\prod_{s\leq i\leq t} A_i} > \prod_{s\leq i\leq s+r} \frac{B_i}{A_i}$
Then,

$$\begin{aligned}
\frac{C}{\prod_{s\leq i\leq t} A_i} &\geq \frac{1}{k} \prod_{s\leq i\leq s+r} \frac{B_i}{A_i} \\
&\geq \frac{1}{1-M^{x-r}} \prod_{s\leq i\leq s+r} \frac{B_i}{A_i} \\
&\geq \frac{1}{1-M^{x-r}} \prod_{s\leq i\leq s+r} (\frac{B_i}{A_i})^2
\end{aligned}$$

Thus, by Lemma 5.1.4, the second path will take c.

5.2 Series-Parallel Graphs

Solving this problem on any series-parallel graph reduces to solving the problem on a chain graph with additional uncrossed edges: We know that the most reliable path is guaranteed to be one of a pair of optimal paths. Thus, we can consider the nodes in this path to be the base of the chain, and the edges in this path to be $a_1 \ldots a_{n-1}$.

We can also simplify the graph by removing all but one of the non-chain paths between each pair of vertices on the most reliable path. If v_i and v_k are vertices on the most reliable path, and there are two paths p_x and p_y between v_i and v_k which contain no other vertices along the chain such that the probability of p_x not failing is higher than that for p_y, then any supposedly optimal pair of paths containing p_y could be improved by replacing it with p_x. This is clearly the case because this change increases the reliability of the second path without increasing the degree to which the two paths are interdependent. Thus, no path containing p_y can be in an optimal pair. That means that all edges only in p_y can be ignored.

Further, if we have one path between v_i and v_k with probability P of not failing, that is equivalent to having one edge between v_i and v_k with weight P. Thus,a series parallel graph reduces to a chain of vertices with single edges connecting pairs of these vertices. Finally, we know that none of these edges could cross because none of the original paths could cross in a series-parallel graph. Thus, this reduces to the problem discussed in the previous section.

5.3 Additional Crossing Edges

Allowing additional edges to cross makes the problem much more difficult. In this case, Lemma 3.0.1 is no longer guaranteed. It is possible that neither of the optimal two paths will be one of the most reliable single paths. Consider the following graph:

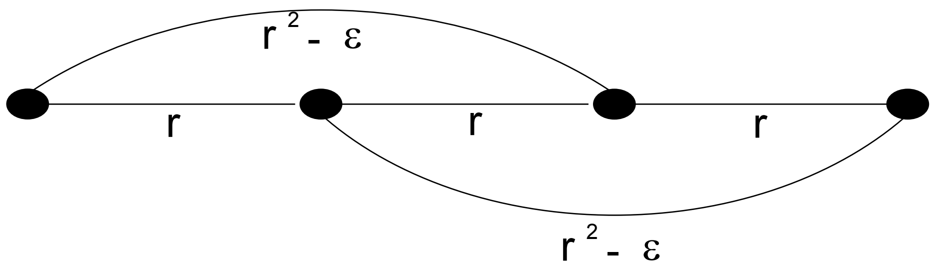

Figure 5.2: This is an example of a non-series-parallel graph for which Lemma 3.0.1 does not hold. The three shorter edges each have length r while the two longer edges have length $r^2 - \epsilon$ each.

Let path $A = \{v_0, v_2, v_3\}$, path $B = \{v_0, v_1, v_3\}$, and path $C = \{v_0, v_1, v_2, v_3\}$. Now,

$P(A) = (r^2 - \epsilon) * r = r^3 - r\epsilon$

$P(B) = r * (r^2 - \epsilon) = r^3 - r\epsilon$

$P(C) = r * r * r = r^3$

Thus, C is the most reliable route. However,

$$\begin{aligned} P(A \vee B) &= (r^2-\epsilon)*r + r(r^2-\epsilon) - (r^2-\epsilon)*r*r(r^2-\epsilon) \\ &= 2r^3 - 2r\epsilon - (r^6 - 2r^4\epsilon + r^2\epsilon^2) \\ &= 2r^3 - 2r\epsilon - r^6 + 2r^4\epsilon - r^2\epsilon^2 \end{aligned}$$

$$\begin{aligned} P(A \vee C) &= (r^2-\epsilon)*r + r*r*r - (r^2-\epsilon)*r*r*r \\ &= 2r^3 - r\epsilon - (r^5 - r^3\epsilon) \\ &= 2r^3 - r\epsilon - r^5 + r^3\epsilon \end{aligned}$$

$$\begin{aligned} P(B \vee C) &= r(r^2-\epsilon) + r*r*r - r*(r^2-\epsilon)*r*r \\ &= 2r^3 - r\epsilon - r^5 + r^3\epsilon \end{aligned}$$

If $\epsilon < \frac{r^5-r^6}{r+r^2+r^3-2r^4}$

$$\begin{aligned} r^5 - r^6 &> (r + r^2 + r^3 - 2r^4)\epsilon \\ 2r^3 - 2r\epsilon - r^6 + 2r^4\epsilon - r^2\epsilon &> 2r^3 - r\epsilon - r^5 + r^3\epsilon \\ 2r^3 - 2r\epsilon - r^6 + 2r^4\epsilon - r^2\epsilon^2 &> 2r^3 - r\epsilon - r^5 + r^3\epsilon \\ 2r^3 - 2r\epsilon - r^6 + 2r^4\epsilon - r^2\epsilon^2 &> 2r^3 - r\epsilon - r^5 + r^3\epsilon \\ P(AorB) &> P(BorC) = P(AorC) \end{aligned}$$

Thus, A and B are the optimal pair of paths, although neither alone is as reliable as C.

Chapter 6

Conclusions

For a restricted class of edge weights, we have been able to solve our problem on a graph with chain topology. For a broader class of edge weights, in some cases the problem can still be solved efficiently. In other cases, we can make approximations and we have shown a lower bound on how good these approximations must be.

In addition, we have seen that solving this problem on the chain graph allows us to solve the problem on any series-parallel graph with similarly restricted edge weights. However, we have shown that the same algorithm will not work on arbitrary acyclic graphs.

Some future areas to consider would include extending this problem to find n paths from a given source to a destination which maximize the probability that at least k arrive at the destination. It is unknown whether it would be possible to extend the current approach for a greater number of paths. In particular, Lemma 3.0.1 has not been proven for n paths.

It would also be extremely useful to be able to solve the problem exactly and efficiently for unrestricted edge weights. As mentioned above, currently only a restricted class of edge weights can be solved exactly in $O(nlogn)$ time, while other edge weights can only be approximated. A simpler but still unsolved problem would be to consider approximations in the case where the ratios $\frac{B_i}{A_i}$ are not distinct.

Finally, we would like to know if there is any way to efficiently solve this problem on a graph with crossing edges, or on an arbitrary acyclic graph. We have shown that the given algorithm will not work in this case. It is possible that some variation may be more successful, or that an entirely different algorithm may be required to solve this more complicated problem.

Bibliography

[1] Ravindra K. Ahuja. Dynamic shortest paths minimizing travel times and costs. citeseer.nj.nec.com/512158.html, 2001.

[2] J. F. Bard and J. E. Bennett. Arc reduction and path preference in stochastic acylic networks. *Management Science*, 37(2):198–215, 1991.

[3] Jonathan F. Bard and Jeanne L. Miller. Probabilistic shortest path problems with budgetary constraints. *Comput. Oper. Res.*, 16(2):145–159, 1989.

[4] Justin A. Boyan and Michael Mitzenmacher. Improved results for route planning in stochastic transportation. In *Symposium on Discrete Algorithms*, pages 895–902, 2001.

[5] A. W. Brander and Mark C. Sinclair. A comparative study of k-shortest path algorithms. In *Proc. 11th UK Performance Engineering Worksh. for Computer and Telecommunications Systems*, September 1995.

[6] I. Chabini. Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time. *Transportation Research Records*, 1645:170–175, 1998.

[7] Kenneth L. Cooke and Eric Halsey. The shortest route through a network with time-dependent internodal transit times. *J. Math. Anal. Appl.*, 14:493–498, 1966.

[8] Amir Eiger, Pitu B. Mirchandani, and Hossein Soroush. Path preferences and optimal paths in probabilistic networks. *Transportation Sci.*, 19(1):75–84, 1985.

[9] David Eppstein. Parallel recognition of series parallel graphs. *Information & Computation*, 98(1):41–55, May 1992.

[10] R. W Hall. The fastest path through a network with random time-dependent travel times. *Transportation Science*, 20(3):182–188, 1986.

[11] D. E. Kaufman and R. L. Smith. Fastest paths in timedependent networks for intelligent vehicle-highway systems applications. *IVHS Journal*, 1(1):1–11, 1993.

[12] Terence Kelly. Further computational results on fastest paths in stochastic time-dependent networks. citeseer.nj.nec.com/26139.html.

[13] A. Kerr. Utility maximising dynamic route selection in acyclic stochastic networks. In *Proceedings of the First Western Pacific and Third Australia-Japan Workshop on Stochastic Models, Christchurch (New Zealand)*, pages 269–277, 1999.

[14] Andrew Kerr. Utility maximising stochastic dynamic programming: An overview. citeseer.nj.nec.com/271595.html.

[15] Chao-Lin Liu and Michael P. Wellman. Using stochastic-dominance relationships for bounding travel times in stochastic networks. citeseer.nj.nec.com/513777.html.

[16] Ronald Prescott Loui. Optimal paths in graphs with stochastic or multidimensional weights. *Comm. ACM*, 26(9):670–676, 1983.

[17] E. D. Miller-Hooks and H. S. Mahmassani. Least possible time paths in stochastic. To appear in Computers and Operations Research. 14, 1997.

[18] I. Murthy and S. Sarkar. Exact algorithms for the stochastic shortest path problem with a decreasing deadline utility function. *European J. Oper. Res.*, 103(1):209–229, 1997.

[19] Ariel Orda and Raphael Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *J. Assoc. Comput. Mach.*, 37(3):607–625, 1990.

[20] Ariel Orda and Raphael Rom. Minimum weight paths in time-dependent networks. *Networks*, 21(3):295–319, 1991.

[21] Stefano Pallottino and Maria Grazia Scutellà. Shortest path algorithms in transportation models: classical and innovative aspects. Technical Report TR-97-06, 14, 1997.

[22] J. Pemberton and R. Korf. Making locally optimal decisions on graphs with cycles. Technical Report, Computer ScienceDepartment, University of California at Los Angeles, 1992., 1992.

[23] Wim Pijls and Antoon Kolen. A general framework for shortest path algorithms. citeseer.nj.nec.com/319172.html.

[24] Daniele Pretolani. A directed hypergraph model for random time dependent shortest paths. citeseer.nj.nec.com/pretolani98directed.html.

[25] K. Scott, G. Pabon-Jimenez, and D. Bernstein. Finding alternatives to the best path. Preprint 970682, The Transportation Research Board, January 1997. From the 76th annual meeting of the TRB., 1997.

[26] Kiseok Sung, Michael G. H. Bell, Myeongki Seong, and Soondal Park. Shortest

paths in a network with time-dependent flow speeds. *European J. Oper. Res.*, 121(1):32–39, 2000.

[27] Michael P. Wellman, Matthew Ford, and Kenneth Larson. Path planning under time-dependent uncertainty. pages 532–539.