

# On the Difficulty of Some Shortest Paths Problems

Amit Bhosle

Department of Computer Science  
University of California, Santa Barbara  
Santa Barbara, CA 93106

Masters of Science

**THESIS DRAFT**

October 1, 2002

# On the Difficulty of Some Shortest Paths Problems

*THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS FOR THE DEGREE OF*

Masters of Science  
*in*  
COMPUTER SCIENCE

*by*  
**Amit Bhosle**

*Under the supervision of*  
**Prof. Subhash Suri**



**Department of Computer Science**  
University of California at Santa Barbara  
Santa Barbara  
CA 93106  
September 2002

## ACKNOWLEDGMENT

I am grateful to my advisor Subhash Suri for his able guidance and encouragement throughout the course of this work. He was equally devoted to this dissertation and provided timely help and motivating discussions which proved invaluable for my research. I would also like to take the opportunity to extend my thanks to Subhash's colleague, John Hershberger, whose insights in the problems proved extremely helpful. Last but not not the least, I thank my wife, parents, brother and sister-in-law for "being there for me" and supporting my decisions throughout the course of this work.

**Amit Bhosle**

Department of Computer Science  
University of California at Santa Barbara  
Santa Barbara  
CA 93106

To my Mom, Baba, Wife, Bhaiya and Bhabhi

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Main Results . . . . .	3
1.2	Related Problems . . . . .	4
1.2.1	$k$ -Pairs Shortest Paths . . . . .	4
1.2.2	$\langle length \rangle \times \langle hops \rangle$ . . . . .	5
1.2.3	Replacement Shortest Paths Tree . . . . .	5
1.2.4	Replacement Paths with Subpaths Deletions . . . . .	6
<b>2</b>	<b>Upper Bounds</b>	<b>7</b>
2.1	Undirected Networks . . . . .	7
2.2	Directed Networks . . . . .	10
<b>3</b>	<b>Lower Bounds</b>	<b>13</b>
3.1	Lower Bound Construction . . . . .	13
3.2	Lower Bound on $k$ -Pairs Shortest Paths . . . . .	16
3.2.1	All Pairs Shortest Paths . . . . .	16
3.2.2	$k$ -Pairs Shortest Paths . . . . .	17
3.3	Limitations of the Path Comparison Model . . . . .	18
3.4	Replacement Paths Lower Bound . . . . .	20
<b>4</b>	<b>Lower Bounds on Related Problems</b>	<b>23</b>
4.1	$k$ -Shortest Simple Paths . . . . .	23
4.2	$\langle Length \rangle \times \langle hop\ count \rangle$ . . . . .	24
4.3	Replacement Shortest Paths Tree . . . . .	27

4.4	Replacement Paths for Subpaths Deletions . . . . .	30
4.4.1	Disjoint Subpaths . . . . .	30
4.4.2	Lower Bound . . . . .	33
4.4.3	Upper Bound . . . . .	36
<b>5</b>	<b>Concluding Remarks and Open Problems</b>	<b>38</b>

# List of Figures

2.1	Unlike the broken path, there exists a $path(v, t)$ which is completely contained in $V_t$ and does not contain the edge $e_i = (v_i, v_{i+1})$ . . . . .	8
2.2	The lemma fails for directed graphs as the path from $v$ to $t$ uses the edge $e$ . . . . .	10
2.3	Candidates for the best replacement path through a node $v \in V_s$ . . . . .	11
3.1	The General Construction for the Lower Bound . . . . .	14
3.2	The Karger-Koller-Phillips Lower Bound Construction : the edges whose weights are modified are the thick ones. . . . .	17
3.3	The lower bound argument breaks on adding superfluous nodes and edges . . . . .	19
4.1	Subpath optimality does not hold for $\langle \text{length} \rangle \times \langle \text{hop count} \rangle$ metric	25
4.2	Modification in the basic construction . . . . .	25
4.3	Construction for Shortest Paths Tree Maintenance Lower Bound . . . . .	28
4.4	Node version of the replacement paths problem for undirected graphs	31
4.5	Disjoint subpaths deletions in undirected graphs . . . . .	33
4.6	Construction for Subpaths Deletions Lower Bound . . . . .	34

## Abstract

We investigate the computational complexity of some shortest path problems in *directed* graphs. The central problem in our study is the *replacement paths* problem: Given a directed graph  $G$  with non-negative edge weights, and a shortest path  $P = \{e_1, e_2, \dots, e_p\}$  between two nodes  $s$  and  $t$ , compute the shortest path distances from  $s$  to  $t$  in each of the  $p$  graphs obtained from  $G$  by deleting exactly one edge  $e_i$ . We prove that this problem has a lower bound of  $\Omega(m\sqrt{n})$  in the worst case whenever  $m = O(n\sqrt{n})$ . This enables us to prove similar bounds on some other problems (eg.  $k$ -pair shortest paths problems,  $k$ -Shortest Paths, and some modified versions of the basic replacement paths problem) closely related to the replacement path problem. All these results hold for the *directed* version of the problem in the *path comparison* model for shortest path algorithms, which forms a natural class of such algorithms including those by Dijkstra and Floyd-Warshall. By contrast, (near) linear time algorithms for some of these problems in *undirected* graphs are already known.

# Chapter 1

## Introduction

Some shortest path problems seem to be more difficult in *directed* graphs than in *undirected* ones in the sense that while the undirected versions of the problems have been solved in near-optimal time, the directed versions have managed to resist efficient solutions. One of the most outstanding examples of such problems is the  $k$ -Shortest Paths problem : Given a directed graph  $G$ , and two nodes  $s$  and  $t$  and an integer  $k$ , one is required to find  $k$  distinct paths from  $s$  to  $t$  in order of increasing weight. Also, the paths are required to be *simple* (loopless). The fastest algorithms for this problem date back to 1971 ([Yen71]) and essentially require  $\Theta(n)$  single source shortest path computations per output path. Since then there have been several “engineering” improvements but the worst case complexity remains  $O(kn(m + n \log n))$  [Yen71, Yen72, Law72]. On the other hand, the undirected version of the problem requires only one single source shortest path computation per output path yielding a worst case time complexity of  $O(k(m + n \log n))$  [HSB01, KIM82]. For the sake of comparison, the version of the problem where the paths need not be simple has been solved to (near) optimality in  $O(m + n \log n + k)$  time by Eppstein [Epp94].

Another problem exhibiting different levels of difficulty in its directed and undirected versions is the *replacement paths* problem. Given a graph  $G$  with non-negative edge weights, a pair of nodes  $s, t$ , and a shortest path  $P = \{e_1, e_2, \dots, e_p\}$  from  $s$  to  $t$ , one needs to compute the shortest path from  $s$  to  $t$  in each of the  $p$  graphs obtained from

$G$  by deleting exactly one edge of  $P$ . (A different version of this problem deals with the deletion of *nodes* on a given shortest path as opposed to edges in this case. Both these versions are known to be computationally equivalent.) The primary application of this problem arises in network routing when new routes need to be computed in response to individual link failures [FT00]. Another motivation comes from computational mechanism design, in which the Vickery-Clarke-Grove scheme is used to elicit true link costs in a distributed but self-interested communication setting, such as the Internet [AT02],[NR99]. In the VCG payment scheme, the bonus to a link agent  $e_i$  equals  $d(s, t, G \setminus e_i) - d(s, t, G|_{e_i=0})$ , where the former is the replacement path length and the latter is the shortest path length from  $s$  to  $t$  with the cost of  $e_i$  set to zero. Computing all these payment values is equivalent to solving the replacement paths problem for  $(G, s, t)$ . The replacement path problem has also been central to most of the algorithms designed for solving the  $k$ -shortest paths problem [HSB01, Yen71, Law72, KIM82]. To see this, notice that the cheapest path among the  $p$  replacement paths is the *second shortest* path from  $s$  to  $t$ . To date, all the known algorithms for the  $k$ -shortest paths use the replacement paths as a subproblem.

A naive algorithm for solving the replacement paths problem runs in  $O(mn + n^2 \log n)$  time, using a single-source-shortest path computation on the graph  $G \setminus e_i$  for  $i = 1, 2, \dots, p$  (note that  $p$  may be as large as  $n - 1$ ). No better algorithm for the directed version of this problem is known. However, as we shall explain later, if  $G$  is undirected, the problem can be solved in asymptotically the same time as one single-source-shortest path computation. The (undirected) version of the problem which deals with node deletions has been solved in the same time bound by Nardeli, Proietti and Widmayer [NPW01] as part of their algorithm for the *most vital node* problem for undirected graphs. This work was based on earlier work by Malik, Mittal and Gupta [MMG89], Ball, Golden and Vohra [BGV89], and Bar-Noy, Khuller and Schieber [BKS95].

There has been extensive work devoted to dynamic maintenance of shortest path distances and/or paths in networks in presence of link failures. Typically, investiga-

tions have focussed on how shortest distances and/or paths may be affected when certain link(s) in the network fails and the queries are of the form  $d(x, y, u, v)$  or  $path(x, y, u, v)$  asking for shortest distance or path from node  $x$  to node  $y$  when the link  $(u, v)$  fails [DT02, DI01, Cho, Kin99, KS99]. Although, single link failures are the only scenario considered, some work has been devoted to multiple link failures [CSC02, MFB99]. The algorithms developed for such problems do not have a very attractive efficiency which may very well be due to the nature of the problem at hand. Some of the older algorithms were nearly as bad as the naive approaches [EG85, Roh85, FMSN98, FMSN00]! The replacement paths problem happens to be a restricted version of such problems. And as we shall see, its undirected version has in fact been solved to near optimality while the dynamic shortest paths maintenance problems have resisted efficient algorithms.

## 1.1 Main Results

We establish a lower bound on the replacement paths problem in directed graphs for any *path comparison* model for shortest paths algorithms proposed by Karger, Koller and Phillips [KKP91]. In this model, an algorithm determines the shortest paths by comparing the lengths of two different paths. Such an algorithm can perform standard operations in unit time, but its access to edge weights is only through the comparison of two paths. In fact, these algorithms are charged a cost proportional only to the number of comparison they make, and not for actually constructing the paths to be compared. Although somewhat restrictive, most of the known shortest paths algorithms fall in this category, including those by Dijkstra [Dij59], Bellman-Ford [Bel58, FF62], Floyd [Flo62], Spira [Spi73], Frieze-Grimmet, and the hidden paths algorithm by Karger, Koller and Phillips [KKP91]. On the other hand, the sub-cubic algorithms (especially those using fast matrix multiplication) by Fredman [Fre76] and [Tak92, Tak95] do not fit this description since they add weights of edges which do not form a path.

However, we discovered another limitation of the path comparison lower bound model,

in the sense that the algorithms are not allowed to add new vertices and/or edges to the input graph since it can invalidate the lower bound arguments. We shall discuss this limitation in greater detail in section (3.3). We thus restrict the scope of our lower bounds somewhat further by not allowing the algorithms to compare paths which cannot fall into the solution space. Again, most of the shortest paths algorithms obey this restriction.

We prove a lower bound of  $\Omega(m\sqrt{n})$  on the replacement paths problem whenever  $m = O(n\sqrt{n})$ . (Put differently, we establish a lower bound of  $\Omega(\min(n^2, m\sqrt{n}))$ .) Consequently, any  $k$ -shortest paths algorithm which uses replacement paths as a subroutine is subject to similar bound. In fact, computing even the *second shortest* simple path in directed graphs has this lower bound. This bound also extends to the version of the replacement paths problem which deals with node deletions rather than edge-deletions. To the best of our knowledge, these are the first non-trivial lower bounds on these problems.

## 1.2 Related Problems

Our lower bound result on the replacement paths problem enables us to establish similar bounds on some other problems which are closely related to the replacement paths problem explained above. Apart from the  $k$ -shortest paths problem, we list some other problems which exhibit similar levels of difficulty.

### 1.2.1 $k$ -Pairs Shortest Paths

The lower bound for the replacement paths problem is based on the result we obtain for the  $k$ -pairs shortest paths problem : Given a directed graph  $G$ , with non-negative edge weights, and  $k$  source-destination pairs  $(s_i, t_i)$ , one needs to compute the shortest path between each  $(s_i, t_i)$  pair. (The sources and destinations need not be unique).

Although this problem has a natural flavor, it has not been studied in its entirety. When  $k = n^2$ , the problem is the famous *All Pairs Shortest Paths* problem which

can be solved in  $O(mn + n^2 \log n)$  time. Karger, Koller and Phillips [KKP91] give a lower bound of  $\Omega(mn)$  for any *path comparison* based algorithm. On the other extreme, we have the single-source-shortest path computation which can be solved in  $O(m + n \log n)$  time. This problem has the trivial lower bound of  $\Omega(m)$ . Thus, almost tight bounds are known for the two ends of the spectrum. However, the remaining part of the spectrum, namely when  $1 < k < n^2$ , has apparently not been studied. We prove a lower bound of  $\Omega(m\sqrt{k})$  for any path-comparison based algorithm for this problem (In our lower bound on the replacement paths problem, we use this result with  $k = n$ ).

### 1.2.2 $\langle \text{length} \rangle \times \langle \text{hops} \rangle$

In their work on Frugal Path mechanism design, Archer and Tardos [AT02] suggest a metric of  $\langle \text{length} \rangle \times \langle \text{hop count} \rangle$  for the replacement paths problem, the motivation being the fact that paths with larger number of edges tend to yield a higher payments to the individual links than those with fewer edges and so favoring paths with fewer hops might work better in practice. This problem is basically the same as the original replacement paths problem, the only difference being the more complex metric. We show that this problem has the same lower bound, i.e.  $\Omega(\min(n^2, m\sqrt{n}))$ .

### 1.2.3 Replacement Shortest Paths Tree

This is a variant of the basic replacement paths problem in which one needs to recompute the entire shortest paths tree of a given source node when an edge of the original shortest path tree is deleted. That is, the difference lies essentially in the fact that now we need to recompute the shortest path to every other node  $v \in G(V)$  from a given source node  $s$  whereas in the original replacement paths problem we needed to do it only for a single destination node. Formally, the problem at hand is :

Given a graph  $G$  with non-negative edge weights and the shortest paths tree  $T_s(G)$  of a node  $s$ , with  $T_s(G) = \{e_1, e_2, \dots, e_{n-1}\}$ , compute the shortest paths tree of  $s$  in each of the  $n - 1$  graphs obtained from  $G$  by deleting exactly one of the edges  $e_i$ .

A naive algorithm for this problem runs in  $O(mn + n^2 \log n)$  time using a single-source shortest path computation for each of the  $n - 1$  graphs obtained from  $G$  by deleting exactly one edge of the original shortest path tree of  $s$ . We show a construction which essentially proves a lower bound of  $\Omega(\text{All-}Pair\text{-Shortest-Paths})$  for a given graph  $G$ . Using the lower bound proof of Karger, Koller and Phillips [KKP91], we arrive at a lower bound of  $\Omega(mn)$  for the directed version of this problem.

### 1.2.4 Replacement Paths with Subpaths Deletions

In this modified version of the basic replacement paths problem, we deal with deletion of *subpaths* of a given shortest path as opposed to deletion of single edges on the shortest path :

Given a graph  $G$  with non-negative edge weights, two nodes  $s$  and  $t$ , a shortest path  $P = \{e_1, e_2, \dots, e_p\}$  from  $s$  to  $t$  and a set  $Q = \{p_1, p_2, \dots, p_k\}$  of  $k$  subpaths of  $P$ , compute the shortest path from  $s$  to  $t$  in each of the  $k$  graphs obtained from  $G$  by deleting exactly one of the given  $k$  subpaths.

A naive algorithm for this problem runs in  $O(k(m + n \log n))$  using a single-source shortest paths computation on each of the  $k$  graphs  $G \setminus p_i$ . Since  $k$  can vary anywhere from 1 to  $n^2$ , this bound can be as large as  $O(mn^2 + n^3 \log n)$ . We prove a lower bound of  $\Omega(m\sqrt{k})$  and an upper bound of  $O(mn + n^2 \log n + kn)$ . We investigate two variants of this particular problem dealing with subpath deletions: (1) when the subpaths are disjoint (in which case the problem can be solved in near-optimal time using a modification of a technique by Nardelli, Proietti and Widmayer [NPW01]) and (2) when the subpaths are overlapping. The lower bounds given above hold even for the case where the subpaths overlap in a very “ordered” fashion forming a staircase-like structure.

# Chapter 2

## Upper Bounds

### 2.1 Undirected Networks

The undirected version of the replacement paths problem can be solved efficiently. Here we present a brief sketch of a (near) optimal algorithm for this problem developed recently by Hershberger and Suri [HS01]. The algorithm is simple and elegant. Its running time is asymptotically the same as that of one single-source shortest paths computation. The algorithm starts out by computing the shortest path trees,  $T_s$  and  $T_t$ , of the source and the destination nodes  $s$  and  $t$  respectively. Furthermore, when an edge  $e_i \in P(s, t)$  is deleted,  $G(V)$  is partitioned into two sets induced by  $T_s$ . Let  $V_s$  and  $V_t$  be the sets containing  $s$  and  $t$  respectively. The key observation is the fact that if a node  $v \in V_t$ , then  $path(v, t)$  cannot contain  $e_i$ , where  $path(v, t)$  denotes the shortest path from  $v$  to  $t$  (that is, the path from  $v$  to  $t$  in  $T_t$ ).

More formally, let  $d(u, v; G)$  denote the shortest path distance from  $u$  to  $v$  in  $G$ . Similarly,  $d(u, v; G \setminus e_i)$  denotes the shortest path distance from  $u$  to  $v$  when edge  $e_i \in G$  is deleted. Note that if  $e_i$  does not lie on the shortest path from  $s$  to  $t$ , then  $d(s, t; G) = d(s, t; G \setminus e_i)$ . Next follows a definition of a *block*. If the shortest path  $P$  from  $s$  to  $t$  is the sequence of vertices  $\{v_1, v_2, \dots, v_k\}$ , in the forest obtained by deleting all the edges of  $P$  from  $T_s$ , the set of vertices connected to  $v_i$  form the block  $B_i$ . The basis of Hershberger and Suri's algorithm [HS01] can be stated in the following lemma:

**Lemma [HS01]:** Let  $v$  be a vertex in  $V_t = \cup_{j=i+1}^k B_j$  for some edge  $e_i = (v_i, v_{i+1})$ . Then  $d(v, t, G) = d(v, t; G \setminus e_i)$ .

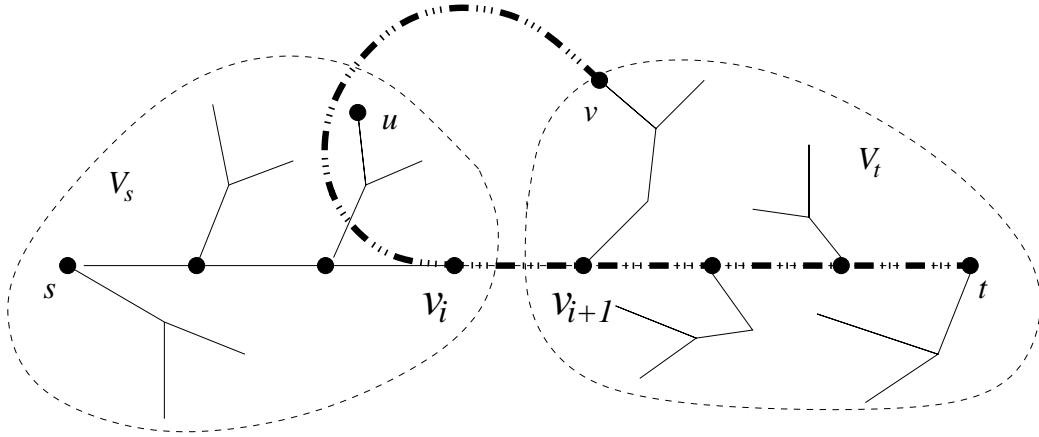


Figure 2.1: Unlike the broken path, there exists a  $path(v, t)$  which is completely contained in  $V_t$  and does not contain the edge  $e_i = (v_i, v_{i+1})$

**Proof:** The proof follows from an easy contradiction: Suppose the shortest path from  $v$  to  $t$ ,  $\pi$ , does contain the edge  $e_i$ , then by subpath optimality, the shortest path from  $v$  to  $v_{i+1}$  is the subpath, say  $\pi_1$ , of this path upto  $v_{i+1}$ . Using the subpath optimality condition on the shortest path from  $s$  to  $v$ , we conclude that the shortest path from  $v_{i+1}$  to  $v$  is the subpath  $\pi_2 = path(v_{i+1}, v)$  of the  $(s, v)$  shortest path. Thus,  $\pi_2 \leq \pi_1$ . Now notice that  $|\Pi| = |\pi - \pi_1 + \pi_2| \leq |\pi|$  and  $\Pi$  does not contain the edge  $e_i = (v_i, v_{i+1})$ .  $\square$

Finally, note that the replacement path for an edge  $e_i \in P(s, t; G)$  must contain exactly one edge crossing the cut induced by  $T_s$  when the edge  $e_i$  is deleted from  $T_s$ . Let  $E_i$  denote the set of edges crossing the cut induced by deleting  $e_i$ . Thus, the goal reduces to computing the following values for each  $e_i$  of the path  $P$  :

$$d(s, t; G \setminus e_i) = \min_{(u,v) \neq e_i, (u,v) \in E_i} \{d(s, u; G \setminus e_i) + weight(u, v) + d(v, t; G \setminus e_i)\}$$

As demonstrated by the above lemma, since the expression on the right side of the above equation remain the same in  $G$  and  $G \setminus e_i$ , it reduces to the following simpler one:

$$d(s, t; G \setminus e_i) = \min_{(u,v) \neq e_i, (u,v) \in E_i} \{d(s, u) + \text{weight}(u, v) + d(v, y)\}$$

An arbitrary way of computing these values may still yield no improvement over the naive approach since there can be  $\Omega(m)$  edges in each set  $E_i$  with as many as  $n - 1$  such  $E_i$ s. An efficient way of computing these values proceeds in a systematic way starting with computation for the replacement path for  $e_1$  and proceeding towards  $e_k$ . Also, notice that  $E_{i+1}$  can be computed quickly using  $E_i$  : simply remove all the edges from  $E_i$  whose *right* end point is  $B_{i+1}$  and insert all edges whose *left* end point is  $B_{i+1}$ . A cost is assigned to each edge  $e = (u, v)$  and is defined as

$$\text{cost}(e) = d(s, u) + \text{weight}(u, v) + d(v, t)$$

To wrap it up, the algorithm begins by maintaining a heap of edges in  $E_1$  with costs as defined above. The minimum value in the heap gives the replacement cost for edge  $e_1$ . At an intermediate step if the heap contains edges of the set  $E_i$  corresponding to the edges crossing the cut induced by deleting  $e_i$ , the minimum value in the heap is the replacement cost for edge  $e_i$ . To update the heap to contain edges in the set  $E_{i+1}$ , those edges whose right end point is  $B_{i+1}$  are deleted and those whose left end point is  $B_{i+1}$  are added. A standard heap results in an  $O(m \log m) = O(m \log n)$  algorithm. A slightly modified algorithm with a more sophisticated heap data structure like the Fredman and Tarjan's Fibonacci Heaps [FT87] results in a time bound of  $O(m + n \log n)$ . The modified algorithm has  $O(n)$  elements in the heap: a representative edge for each of the  $O(n)$   $s - t$  cuts. Whenever a new candidate edge appears, its cost is compared to the cost of the edge representing its cut in the heap. If the new edge offers a shorter path, a *DecreaseKey* operation is performed. In this implementation, there are  $O(m)$  *DecreaseKey* operations but only  $O(n)$  inserts/deletes and *findMin* operations. Only the delete operation costs  $O(\log n)$  while all others take constant time (amortized) each. Thus, the claimed bound of  $O(m + n \log n)$  is achieved.

## 2.2 Directed Networks

The difficulty in the directed version of the problem arises because the lemma stated above for the undirected graphs does not hold in case of directed ones. See figure (2.2) for an example where it fails.

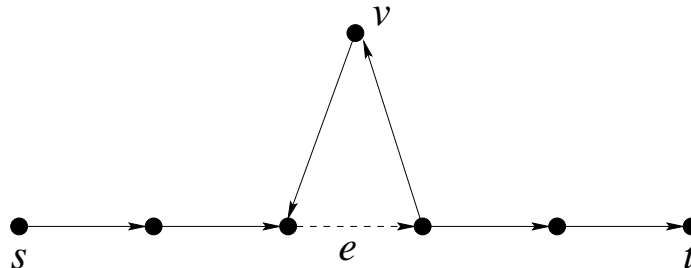


Figure 2.2: The lemma fails for directed graphs as the path from  $v$  to  $t$  uses the edge  $e$

The naive approach, as stated earlier, requires  $O(mn + n^2 \log n)$  time using a single source shortest path computation for after deleting each edge (one at a time) on the given shortest path from  $s$  to  $t$ . We present here an algorithm which requires  $O(APSP)$  time, that is, it solves the replacement paths problem using time dominated by one *All Pairs Shortest Paths* computation as opposed to  $O(n)$  single source shortest path computation : while the latter may be cubic for dense graphs, subcubic algorithms have been designed for the former using fast matrix multiplication [Fre76, Tak92, Tak95] with the fastest among them running in time  $O(n^3 \sqrt{\log \log n / \log n})$ . See [GM97a, GM97b, Sei95, SZ99] for other algorithms for the *APSP* problem with various restrictions. If the heaviest edge in the graph has weight  $C$ , the *APSP* problem can be solved in  $O(C^{.681} n^{2.575})$  [Zwi98]. Using the above algorithm for replacement paths, the  $k$ -shortest paths problem can be solved in  $O(k \cdot APSP)$  time, which would be an improvement for dense graphs with small integer weights. The approach is very much similar to the upper bound result of  $O(m\sqrt{n})$  as mentioned in [HSB02].

We start off by computing the shortest path tree,  $T_s$ , of  $s$  (if not already given)

and the All Pairs Shortest Paths in the graph  $G'$  obtained from  $G$  by deleting all the edges lying on the shortest path from  $s$  to  $t$ . Let us denote the path from a node  $v$  to  $u$  in  $G'$  by  $p_{G'}(v, u)$  and the corresponding path in  $G$  by  $p_G(v, u)$ . Note that deleting any edge on  $p_G(s, t)$  splits  $T_s$  in two components. In this cut, denote by  $V_s$  the component containing  $s$  and the other by  $V_t$ . Consider the case when an edge  $e_i \in p_G(s, t)$  is deleted (See figure (2.3)).

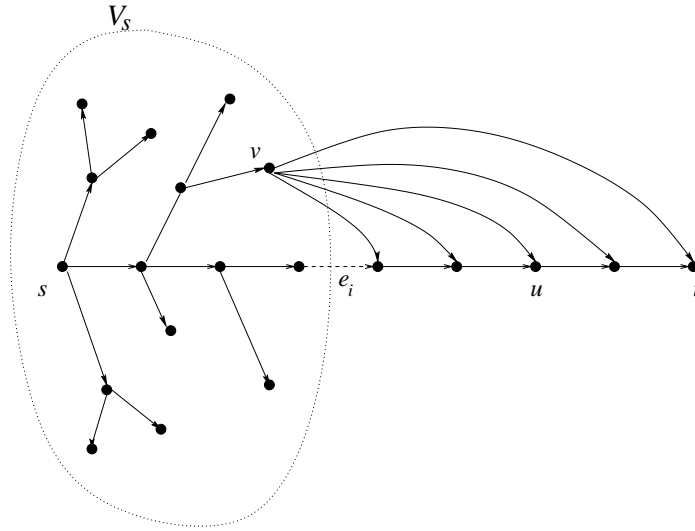


Figure 2.3: Candidates for the best replacement path through a node  $v \in V_s$

The weight of the path through a node  $v \in V_s$  which touches the spine  $p_G(s, t)$  at a node  $u$  is  $d_G(s, v) + d_{G'}(v, u) + d_G(u, t)$  where  $d_{G'}$  and  $d_G$  represent the weights of the paths  $p_{G'}$  and  $p_G$  respectively. The first and third terms in the above expression are obtained from  $T_s$  while the second one was computed in the APSP computation. Obviously, the replacement path for the edge  $e_i$  would follow the path in  $T_s$  to some node  $v$  before using an edge across the cut and joining the spine at some node  $u$ . If  $p_G(s, t) = \{s = u_0, u_1, u_2, \dots, u_{k-1}, u_k = t\}$ , the best replacement path passing through a node  $v \in V_s$  would be :

$$path_v(s, t) = \text{MIN}_{j=i+1}^k \{d_G(s, v) + d_{G'}(v, u_j) + d_G(u_j, t)\} \text{ with } e_i = (u_i, u_{i+1}). \quad (2.1)$$

Note that the paths involved in the above minimization *do not* use the edge  $e_i$ , owing to the construction of  $G'$ . Finally, the best replacement path for the edge  $e_i$  would be a minimum of the replacement paths through the nodes in  $V_s$ . That is,

$$path_{G \setminus e_i}(s, t) = \text{MIN}_{v \in V_s} path_v(s, t) \quad (2.2)$$

Again, as the case with the algorithm by Hershberger and Suri [HS01], an arbitrary way of computing these values may not result in much of an improvement. We start in an orderly fashion, but from the tail of  $p_G(s, t)$  rather than the head as in [HS01]: the reason being that this allows updating the  $path_v(s, t)$  values more easily and quickly. When computing the replacement path for  $e_i$ , we update this value for all  $v \in V_s$  as  $path_v(s, t) = \min\{path_v(s, t), d_G(s, v) + d_{G'}(v, u_{i+1}) + d_G(u_{i+1}, t)\}$  where the first term on the r.h.s. is the present best value for  $path_v(s, t)$ .

The time complexity is straight forward:  $|V_s|$  is at most  $O(n)$  for any deleted edge  $e_i \in p_G(s, t)$ . Updating the  $path_v(s, t)$  values requires constant time per node  $v \in V_s$ , per deleted edge  $e_i \in p_G(s, t)$ . Since  $|p_G(s, t)|$  can also have  $O(n)$  edges, we arrive at a time complexity of  $O(n^2)$  after the initial APSP computation. And since any APSP algorithm requires  $\Omega(n^2)$  time, the overall complexity of the algorithm is dominated by the APSP computation.

# Chapter 3

## Lower Bounds

In this section, we shall discuss our lower bound result on the replacement paths problem. Also included in this section is our lower bound result on the  $k$ -pairs shortest paths ( $kPSP$ ) problem which we use for the former. The  $k$ -pairs shortest path lower bound is in turn based on the all pairs shortest paths lower bound by Karger, Koller and Phillips [KKP91].

### 3.1 Lower Bound Construction

To establish our main result on the lower bound of replacement paths problem, we reduce the problem of computing the replacement path for an edge on the shortest path to computing the shortest path between a given source and destination node between another graph. More specifically, we reduce the problem of computing an instance of an  $n$ -pairs shortest paths problem ( $NPSP$ ) to the replacement paths problem. We describe the general construction below.

We construct a directed path  $P = (s = v_0, v_1, \dots, v_n = t)$  with the  $cost(v_i, v_{i+1}) = 0, \forall 0 \leq i \leq n - 1$ . Consider an instance of the  $n$ -pairs shortest paths problem with  $H$  as the directed weighted graph with  $m$  edges and  $n$  vertices, and  $n$  source-destination pairs  $(s_i, d_i), \forall 1 \leq i \leq n$ . Let  $w$  be the heaviest edge in  $H$  and let  $W = nw$ . We map the edges of  $P$  to the specified source-destination pairs of the  $NPSP$  instance and

introduce edges to connect the vertices on  $P$  to those of  $H$  as follows : If the edge  $(v_i, v_{i+1})$  is mapped to the pair  $(s_j, d_j)$  of  $H$ , create the following edges :

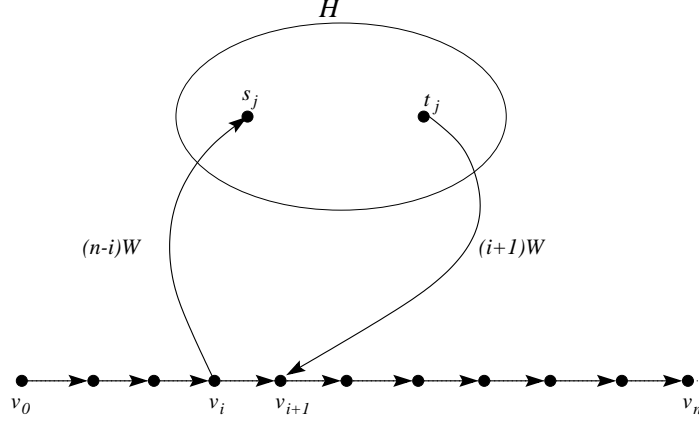


Figure 3.1: The General Construction for the Lower Bound

- Edge directed from  $v_i$  to  $s_j$  with  $cost(v_i, s_j) = (n - i)W$ .
- Edge directed from  $d_j$  to  $v_{i+1}$  with  $cost(d_j, v_{i+1}) = (i + 1)W$ .

See the figure (3.1) for an illustration. This new graph has  $O(n)$  vertices and  $O(m)$  edges and satisfies the following property:

**Lemma :** *The replacement path for the edge  $(v_i, v_{i+1})$  follows the spine till  $v_i$ , enters the graph  $H$  at  $s_j$ , exits it at  $d_j$ , and joins the spine back at  $v_{i+1}$  (where the edge  $(v_i, v_{i+1})$  is mapped to the pair  $(s_j, d_j)$ ). Then, the replacement path has weight*

$$(n + 1)W + d_H(s_j, d_j)$$

where  $d_H(s_j, d_j)$  is the shortest path distance from  $s_j$  to  $d_j$  in the graph  $H$ .

**Proof:** The proof follows from some simple observations about the quantity  $W$ . This value is larger than any simple path in  $H$ , which can be at most  $(n - 1)w$  where  $w$  is the heaviest edge in  $H$ . Thus, if the replacement path for an edge  $(v_i, v_{i+1})$  leaves the spine before  $v_i$ , say  $v_k$  with  $k < i$ , then the possible gain while traversing through  $H$  is more than offset by the huge difference in the weights of the edges leaving  $v_k$  and  $v_i$ .

Thus, the replacement path follows the spine as long as it possibly can. Similarly, it joins the spine at  $v_{i+1}$  since, again the huge difference in the edges at  $v_{i+1}$  and some  $v_l$  with  $l > i+1$  forbids it to do otherwise. For a more mathematical proof, let us consider the weight of a replacement path which leaves the spine at some vertex  $v_k$  with  $k < i$ . This path has weight at least  $(n-k)W + (i+1)W \geq (n+2)W > (n+1)W + d_H(s_j, t_j)$  where the last expression is the weight of the path which leaves and joins the spine  $P$  at  $v_i$  and  $v_{i+1}$  respectively. Investigating the weight of paths joining the spine after  $v_{i+1}$  also proves that such paths are heavier and cannot be the replacement paths.  $\square$

The above construction states that a solution for the replacement paths problem implies a solution for the NPSP problem. Using a lower bound construction for the latter, we arrive at the lower bound for the former. However, for the lower bound to follow directly, we need to impose certain restrictions on the class of algorithms we are dealing with. As stated earlier, the bound holds for *path comparison based* algorithms. That is, the algorithms are allowed only to compare the sums of the weights of edges which form a *path* in the graph and not any arbitrary set of edges. This is a natural model and includes most of the known shortest paths algorithms - Floyd, Dijkstra, Spira and the Hidden Paths algorithm in [KKP91]. However, this excludes the algorithms using fast matrix multiplication [Fre76, Tak92].

**Definition :** *A path comparison model lower bound for graph problems is path addition insensitive if addition of extra nodes and edges to the input graph does not affect the solution or disrupt the lower bound argument.*

From the general construction discussed in this section showing the reduction of an  $n$ -pairs shortest paths problem instance to the replacement paths problem, we have established the following theorem :

**Theorem :** *If there is a directed weighted graph  $H$  with  $O(n)$  vertices and  $O(m)$  edges, and with  $n$  specified source-destination pairs such that any algorithm must spend at least  $\Omega(f(m, n))$  time, then the same lower bound applies to the replacement*

*paths problem provided that the lower bound argument is sufficiently robust.*

## 3.2 Lower Bound on $k$ -Pairs Shortest Paths

We now discuss the lower bound proof for the  $k$ -pairs shortest paths problem which is just a minor variation of the *all pairs shortest paths* lower bound by Karger-Koller-Phillips [KKP91].

### 3.2.1 All Pairs Shortest Paths

The construction involves a directed weighted complete tripartite graph. The vertices in the three columns are named  $x_a$ ,  $y_b$  and  $z_c$ . For the All Pairs Shortest Paths lower bound as described in [KKP91], the indices of the vertices range from 0 to  $n - 1$  each, producing  $\Theta(n^3)$  triples. The algorithm is required to compute the  $x_a$  to  $z_c$  distances for  $0 \leq a, c \leq n - 1$ . The argument depends on the fact that if an algorithm fails to consider any  $x - y - z$  triple, say  $x_{a^*}$ ,  $y_{b^*}$  and  $z_{c^*}$ , then the weights of the edges can be modified in such a way that :

- $(x_{a^*}, y_{b^*}, z_{c^*})$  becomes the shortest path from  $x_{a^*}$  to  $z_{c^*}$ .
- The relative ordering of all other paths remains intact.

This forces the algorithm to include all the  $\Theta(n^3)$  paths in the computation, thus enforcing the said lower bound. If the number of edges is less than  $O(n^2)$ , the middle column has only  $m/2n$  vertices, that is, the index  $b$  ranges from 0 to  $m/2n$ , thus producing  $\Theta(mn)$  paths for the algorithm to investigate.

For the sake of completeness, we mention the precise weights and their modifications for the above construction. The weights assigned to the edges initially are of the following form :

$$||x_a, y_b|| = [1, 0, a, 0, b, 0]_{n+1} \quad (3.1)$$

$$||y_b, z_c|| = [0, 1, 0, c, 0, -b]_{n+1} \quad (3.2)$$

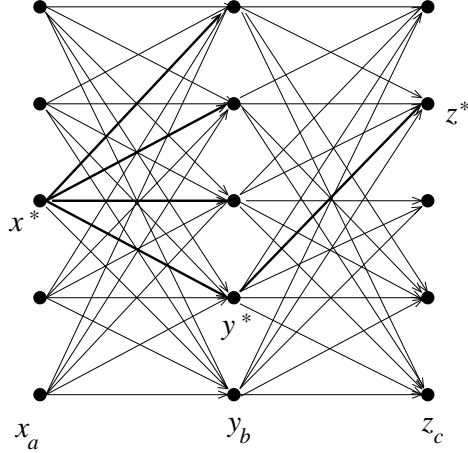


Figure 3.2: The Karger-Koller-Phillips Lower Bound Construction : the edges whose weights are modified are the thick ones.

where the right hand side is a 7-bit number in base  $n + 1$ , defined as  $[\phi_r, \dots, \phi_1, \phi_0]_\alpha = \sum_{i=0}^r \phi_i \alpha^i$ . The negative bits do not create a problem since a higher order positive bit ensures that the entire number remains positive. Moreover, the graph is a *DAG* and there cannot be any negative cycles.

If the algorithm ignores a triple  $(x_{a^*}, y_{b^*}, z_{c^*})$ , the weights are modified such that the two conditions mentioned above hold :

$$\|x_{a^*}, y_b\| = [1, 0, a^*, 0, 0, b, b]_{n+1} \quad \forall b \leq b^* \quad (3.3)$$

$$\|y_{b^*}, z_{c^*}\| = [0, 1, 0, c^*, 0, -b^*, -n]_{n+1} \quad (3.4)$$

It is easy to verify that after this modifications, the ignored path becomes the shortest path between  $x_{a^*}$  and  $z_{c^*}$  while all other paths maintain their relative order, and the reader is referred to [KKP91] for the details.

### 3.2.2 $k$ -Pairs Shortest Paths

A minor modification allows us to establish a lower bound of  $\Omega(\min(n^2, m\sqrt{k}))$  for the  $k$ -pairs shortest paths problem for any value  $1 \leq k \leq n^2$  : the indices  $a$  and  $c$

range from 0 to  $\sqrt{k}$  each, while the index  $b$  varies from 0 to  $\min(n, m/\sqrt{k})$ . Thus, as long as  $m = O(n\sqrt{k})$ , the total number of vertices in the graph is  $O(n)$ , the number of edges is  $O(m)$  and the number of triples for the algorithm to investigate is  $O(m\sqrt{k})$ . Any path comparison based algorithm must investigate all these triples to compute all the  $x_a, z_c$  distances correctly.

**Theorem:** *For any  $n, k$  and  $m$ , with  $1 \leq k \leq n^2$  and  $m = O(n\sqrt{k})$ , there exists a directed graph with  $n$  nodes,  $m$  edges, and  $k$  source-destination pairs  $(s_i, t_i)$ , such that any path comparison based algorithm must spend  $\Omega(m\sqrt{k})$  time computing the shortest path distances for the  $k$  pairs  $(s_i, t_i)$ .*

### 3.3 Limitations of the Path Comparison Model

In this section, we discuss some of the limitations of the Karger-Koller-Phillips [KKP91] lower bound construction which is (most likely) inherent to the path-comparison model for shortest path algorithms proposed by them in [KKP91]. Although the argument presented by them is clean and straight forward, one important *implied* restriction is that the algorithms are not allowed to add nodes and vertices to the input graph, since as we shall show, this invalidates the basic argument. The argument is based on the fact that if an algorithm ignores a triple  $(x_{a^*}, y_{b^*}, z_{c^*})$  of vertices in the tripartite graph, then by modifying the edge weights, (1) this ignored path can be made the shortest path between  $x_{a^*}$  and  $z_{c^*}$ , and (2) the relative order among all other paths is preserved. We show that by adding superfluous nodes and edges to the input graph, condition (2) no longer holds after the modification. Consider the graph  $H'$  constructed from the tripartite graph  $H$  used in the lower bound construction by adding two nodes  $s$  and  $t$ , and edges of weight 0 from  $s$  to all the (source) vertices in the first column, from all the (destination) vertices in the third column to  $t$ , and an edge of weight  $W = nw$  (where  $w$  is the heaviest edge in  $H$ ) from  $t$  to  $s$ . See figure (3.3).

Notice that in the strongly connected graph  $H'$ , all the  $(x_a, z_c)$  shortest distances and paths are identical to those in  $H$ . Furthermore, even though it is far from obvious

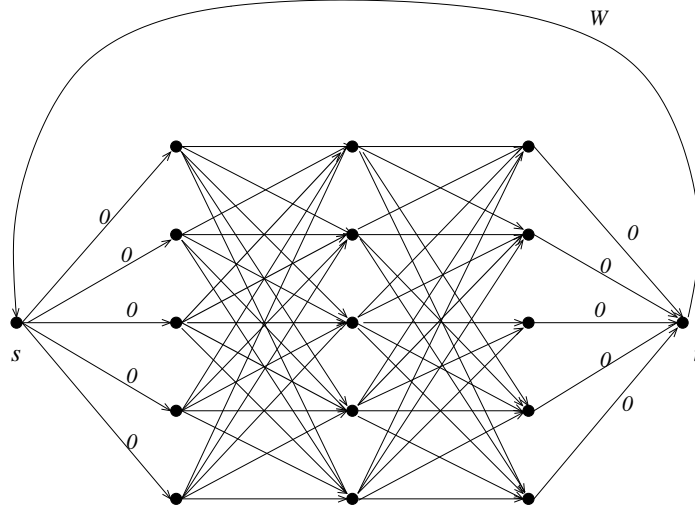


Figure 3.3: The lower bound argument breaks on adding superfluous nodes and edges

how the addition of the extra nodes and edges can make it easier to compute the shortest paths, the basic argument of [KKP91] fails to hold, since as we shall see, the relative order of some of the paths *does* change.

**Lemma :** *The Karger-Koller-Phillips lower bound model for path comparison based algorithms, as presented in [KKP91], is not path addition insensitive.*

**Proof :** Suppose that the weights are modified to make the path  $(x_{a^*}, y_{b^*}, z_{c^*})$  the shortest path from  $x_{a^*}$  to  $z_{c^*}$ . Now for  $3 < b^* < n - 1$  and  $a^* \neq 0$ , consider the following two paths:

$$\pi_1 = (x_{a^*}, y_3, z_0, \dots, \text{loop} \dots, x_0, y_{n-2}, z_1)$$

$$\pi_2 = (x_{a^*}, y_0, z_0, \dots, \text{loop} \dots, x_0, y_{n-1}, z_1)$$

In the original graph,  $\|\pi_1\| > \|\pi_2\|$  since the weights of these paths are

$$\|\pi_1\| = [2, 2, a^*, 1, n + 1, -(n + 1), 0]_{n+1} + W$$

$$\|\pi_2\| = [2, 2, a^*, 1, n - 1, -(n - 1), 0]_{n+1} + W$$

However, after the weights are modified, this order reverses since now,

$$||\pi_1|| = [2, 2, a^*, 1, n - 2, -(n - 2), 3]_{n+1} + W$$

$$||\pi_2|| = [2, 2, a^*, 1, n - 1, -(n - 1), 0]_{n+1} + W$$

and thus,  $||\pi_1||' < ||\pi_2||'$ . Notice that this reversal happens even though neither of these paths contain the edges  $(x_{a^*}, y_{b^*})$  or  $(y_{b^*}, z_{c^*})$ . This completes the proof.  $\square$

Another possible weakness of the lower bound argument might be that it charges an algorithm just for comparisons between two paths and not for their construction. In fact, the algorithm can compare arbitrarily long paths at the same cost as comparing two edges ! This provides scope for an algorithm to consider a long non-simple path which encodes all the relevant paths as subpaths at unit cost. Just as the unbounded word-size RAM model can be (theoretically) abused to sort large numbers in linear time [KR84, PS80], it might be possible to use such long non-simple paths to solve such shortest paths problems much faster.

### 3.4 Replacement Paths Lower Bound

We discuss our main result in this section, which is the lower bound on the replacement paths. In order to establish this bound, we use the result on the  $k$ -pairs shortest paths lower bound of  $\Omega(\min(n^2, m\sqrt{k}))$  with  $k = n$ . Had the Karger-Koller-Phillips lower bound construction been path addition insensitive, it would have established our bound directly. But unfortunately, as discussed in the previous section, it is not so, thus forcing us to limit the scope of our lower bound somewhat. We restrict ourselves to consider only *single-detour* path comparison based algorithms as defined below :

**Definition:** *Given a path  $\Pi$ , and shortest path  $P$  from  $s$  to  $t$  in  $G$ , a detour is any subpath of  $\Pi$  which begins and ends on  $P$ , but does not contain any edges of  $P$ .*

Consequently, a single-detour path comparison based algorithm is one which investigates paths with at most one detour for solving the replacement paths problem. This is reasonable since, as shown below, a path with multiple detours cannot be part of the solution.

**Lemma:** *Given a directed graph  $G$ , and an instance of the replacement paths problem with  $P = (e_1, e_2, \dots, e_p)$  as the shortest path from  $s$  to  $t$ , if  $P_i$  is the replacement path for the edge  $e_i$ , then  $P_i$  cannot have more than one detour.*

**Proof:** The proof is based on the subpath optimality of the shortest paths. A detour which creates a non-simple path can be ignored since the graph does not contain negative cycles and eliminating the cycle would only make the path cheaper. If a simple path has two detours, at most one of them bypasses  $e_i$ , and the other can be eliminated by following  $P$  between its end points. Subpath optimality ensures that this resulting path is shorter than the original one with two detours. This completes the proof.  $\square$

All the known algorithms for the replacement paths do obey the above restrictions. Also, all of the known algorithms for problems related to shortest paths do exploit the local optimality in the sense that if an algorithm has determined that among two paths  $\pi_1$  and  $\pi_2$  between  $u$  and  $v$ ,  $\|\pi_1\| < \|\pi_2\|$ , and both of them satisfy any other criteria imposed by the problem, it would never consider  $\pi_2$  when considering paths including a  $u - v$  subpath. By allowing the algorithms to include only single detour paths, we disallow such useless comparisons.

We are now ready to prove the lower bound on the replacement paths problem. We use the generic construction specified in figure (3.1) specialized for the graph  $H$  involved to be the graph in the  $k$ -pairs shortest paths lower bound construction with  $k = n$ .

**Lemma:** *Any single-detour path-comparison based algorithm for the replacement*

paths must investigate all the triples of the form  $(x_a, y_b, z_c)$  in the subgraph  $H$  of  $G$ . If not, we can modify the edge weights in  $H$  such that the replacement paths with the ignored triple as its subpath becomes the shortest replacement path while all other (single-detour) paths preserve their relative order.

**Proof:** Assume that the algorithm ignores the triple  $(x_{a^*}, y_{b^*}, z_{c^*})$ . We modify the edge weights as stated in equations (3.3),(3.4). It is easy to see that the replacement path passing through these three vertices becomes the shortest replacement path. Now let us investigate the relative order of other (single-detour) paths. If the paths use the same *connecting edge* (those introduced to connect  $P$  to  $H$ ), the argument by Karger-Koller-Phillips [KKP91] proves that their order remains intact. In case they use different connecting edges, their relative order is determined solely by the weights of these edges since these weights are much larger than any path in  $H$ .  $\square$

We have thus proved our main theorem :

**Theorem:** *For any  $n$  and  $m$  with  $m = O(n\sqrt{n})$ , there exists a directed weighted graph  $G$  with  $O(n)$  nodes and  $O(m)$  edges, and two specified vertices  $s$  and  $t$ , such that any single-detour path-comparison based algorithm for solving the replacement paths problem on  $G$  must spend at least  $\Omega(m\sqrt{n})$  time.*

A slight modification to our main construction enables us to establish a similar bound for the *node* version of the replacement paths problem : we introduce a node in the middle of each edge of  $P$ . The replacement path for such nodes is exactly the same as the replacement path of the corresponding original edges.

# Chapter 4

## Lower Bounds on Related Problems

### 4.1 $k$ -Shortest Simple Paths

We mentioned the problem of finding  $k$ -shortest simple paths between two specified nodes  $s$  and  $t$  in a given directed weighted graph  $G$ . The fastest algorithm for the problem date back to Yen [Yen71] in 1971. Using modern data structures, its worst case time complexity is  $O(kn(m + n \log n))$ . It essentially performs  $\Theta(n)$  single source shortest paths computations per output path. On the other hand, the undirected version of the problem requires only  $O(1)$  single source shortest paths computation per output path yielding a time complexity of  $O(k(m + n \log n))$ .

The various algorithms for the said problem, including those by Yen, and later by Lawler [Law72], Katoh [KIM82] and several heuristic improvements to Yen's algorithm [BS95, HC99, MP00, MPS97, Per86], work by solving multiple instances of the replacement paths. The basic idea is that the  $i^{\text{th}}$  shortest path must differ from each of the  $i - 1$  shorter paths by at least one edge, and thus new candidates are generated by solving the replacement paths problem for them. Our lower bound on the replacement paths implies that any algorithm for the  $k$ -shortest paths which uses replacement paths as a subroutine is also subjected to the  $\Omega(\min(n^2, m\sqrt{n}))$  lower

bound. In fact, even computing the *second* shortest path is subject to this bound. Thus, any improvement in the algorithms for the  $k$ -shortest paths problem can be achieved only if the algorithm finds a clever way of finding the shortest replacement path without computing all of them ! That is, it should compute  $\min_i d(s, t; G \setminus e_i)$  without computing all  $d(s, t; G \setminus e_i)$ .

## 4.2 $\langle \text{Length} \rangle \times \langle \text{hop count} \rangle$

In their work on Frugal Path Mechanisms, Archer and Tardos [AT02] suggest a new metric for the replacement paths, namely the *product* of the *length* and the *number of hops* of the path. The motivation is based on a negative result proved in their paper : any reasonable mechanism that motivates the owners of the links to bid truthfully is bound to pay a huge premium in the worst case. The situation becomes worse when the involved paths have a large number of edges since the premium is proportional to the number of hops in the paths. Traditionally, in many situations, if two paths have the same weight, the one with fewer edges is preferred. Even Dijkstra's shortest path algorithm breaks ties in favor of paths with fewer edges. The  $\langle \text{length} \rangle \times \langle \text{hop count} \rangle$  metric is another way of favoring paths with fewer edges. When using the Vickery-Clark-Groves auction mechanism, one needs to compute the payments to all the links that are in a winning shortest path. These payments correspond exactly to determining the replacement paths in  $G \setminus e_i$ , where  $e_i$  is an edge of the winning shortest path.

It is interesting to note that the  $\langle \text{length} \rangle \times \langle \text{hop count} \rangle$  metric does not respect the subpath optimality condition. See figure (4.1) for such an example. In the example shown,

$$\|\pi_1\| = \|\{s, a, b, c\}\| = (10 + 20 + 100) \times 3 = 390$$

$$\|\pi_2\| = \|\{s, b, c\}\| = (70 + 100) \times 2 = 340$$

$$\|\pi_3\| = \|\{s, a, b\}\| = (10 + 20) \times 2 = 60$$

$$\|\pi_4\| = \|\{s, b\}\| = 70 \times 1 = 70$$

The optimal path from  $s$  to  $c$ ,  $\pi_2$ , has  $\pi_4$  as a subpath, but  $\pi_4$  is *not* the optimal path

from  $s$  to  $b$ . The optimal path from  $s$  to  $b$  is  $\pi_3$ .

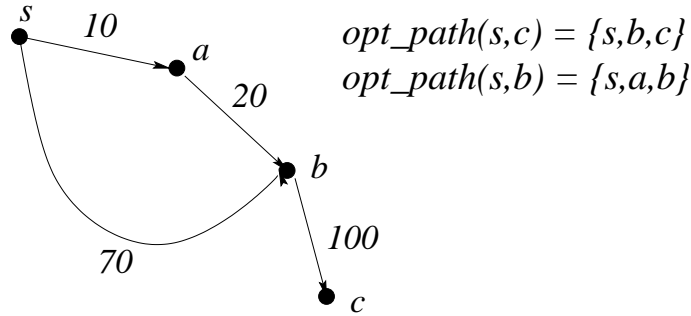


Figure 4.1: Subpath optimality does not hold for  $\langle \text{length} \rangle \times \langle \text{hop count} \rangle$  metric

However, using a slight modification in our lower bound construction, we can get around this problem. We append a path of  $2n$  zero weight edges to the shortest path spine  $P$  in the original replacement paths lower bound construction and move  $t$  to the end of this new appended path  $P$ . This modified graph,  $G'$ , still has  $O(n)$  vertices and  $O(m)$  edges. Note that there are no replacement paths for the newly introduced edges and we are now interested only in the replacement paths for the original set of edges, numbered from  $e_1, \dots, e_n$ . See figure (4.2).

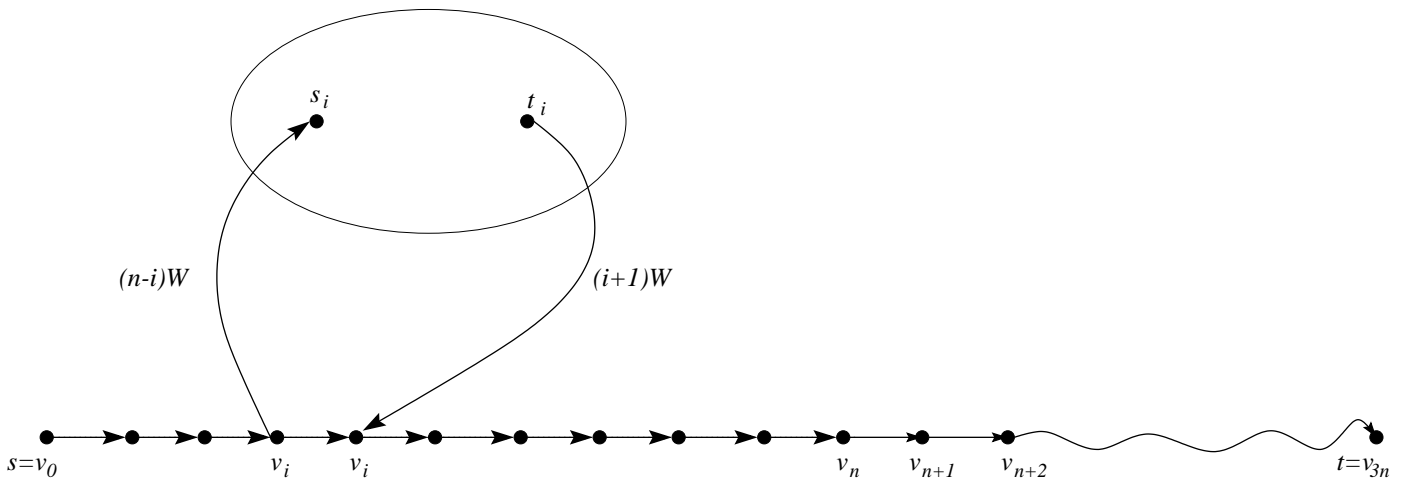


Figure 4.2: Modification in the basic construction

**Lemma:** *In the graph  $G'$  constructed above, the replacement paths for the edge  $e_i$  for  $i=1,2,\dots,n$ , based on the metric  $\langle \text{length} \rangle \times \langle \text{hop count} \rangle$ , has exactly one detour. Furthermore, the replacement path for the edge  $e_i$  has weight*

$$(3n + 3)((n + 1)W + d_H(s_j, t_j))$$

where the edge  $e_i = (v_i, v_{i+1})$  is mapped to the pair  $(s_j, t_j)$  in  $H$ .

**Proof:** The proof is based on the observation that a path making  $d$  detours, with detour  $r$  skipping  $k_r$  edges of  $P$ , has weight exactly equal to

$$(3n - \sum_r k_r + 4d) \times \left\{ (dn + \sum_r k_r)W + \sum_r l_r \right\}$$

where the first term is the number of hops in the path and the second term is its weight ( $l_r < W$  is the weight of the subpath in  $H$ ). For the expression for the number of hops,  $3n$  is the total number of edges on  $P$ . Further, notice that each detour includes 4 extra edges not on  $P$  (2 inside  $H$  and the 2 connecting edges). Thus, the total number of edges is  $3n + 4d - (\text{number of edges of } P \text{ skipped})$ . If a detour  $r$  leaves  $P$  at  $v_i$  and joins at  $v_j$ , the total weight of the 2 connecting edges is  $(n + j - i)W$ . Following our assumption that detour  $r$  skips  $k_r$  edges of  $P$ , the weight of its connecting edges is  $(n + k_r)W$ . Thus, the total weight of the detour  $r$  is  $(n + k_r)W + l_r$ .

As seen from the above expression, it grows quadratically with  $d$  and is minimized only if  $d = 1$ , that is, the path has exactly one detour. Furthermore, a single detour path  $\pi$  skipping  $k$  edges has  $(3n - k + 4)$  hops, and its length satisfies  $(n + k)W < \|\pi\| < (n + k + 1)W$ . Again, the  $\langle \text{length} \rangle \times \langle \text{hop count} \rangle$  metric for  $\pi$  is minimized if  $k = 1$ . This completes the proof.  $\square$

Thus, the replacement paths problem with the modified metric is also subject to the same lower bound, namely  $\Omega(\min(n^2, m\sqrt{n}))$ .

### 4.3 Replacement Shortest Paths Tree

In this section discuss a slight variation of the replacement paths problem which deals with computing the entire shortest paths tree as opposed to just the shortest path to one specified destination node. More formally, given a directed weighted graph  $G$ , a source node  $s$ , and the shortest path tree,  $T_s$  of  $s$ , where  $T_s$  contains the edges  $\{e_1, e_2, \dots, e_{n-1}\}$ , we are required to compute the shortest path tree in each of the  $n - 1$  graphs obtained from  $G$  by deleting exactly one edge of  $T_s$ . A naive approach would take  $O(mn + n^2 \log n)$  time using a single source shortest path computation per deleted edge of the tree. We establish a lower bound of  $\Omega(mn)$  (in the path-comparison model) for the directed version of this problem implying that the naive approach is in fact optimal for graphs with  $m = \Omega(n \log n)$ .

See figure (4.3). The lower bound construction involves a spine  $P = \{s = p_0, p_1, \dots, p_n\}$  of  $n$  edges, each of weight 0. As in our replacement paths lower bound construction, we reduce a given All-Pair-Shortest-Paths problem to this problem. Again, if  $H$  is an arbitrary graph on  $n$  vertices and  $m$  non-negatively weighted edges, we choose  $W = nw$ , where  $w$  is the heaviest edge in  $H$ . For each vertex  $v_i \in H(V)$ , we have a corresponding vertex  $p_i$  on our spine  $P$  ( $1 \leq i \leq n$ ) and we introduce edges directed from each  $p_i$  to the corresponding  $v_i$  with weight given by :

$$\|p_i, v_i\| = (n - i)W$$

**Lemma:** *The original shortest path tree of  $s = p_0$  contains the shortest path tree of  $v_n$  in  $H$  as a subtree. In other words, shortest path from  $s$  to any node  $v_i \in H$  follows  $P$  to  $p_n$  and then enters  $H$  at  $v_n$ , followed by a shortest path from  $v_n$  to  $v_i$ . Furthermore, the shortest path tree of  $s$  in the graph  $G \setminus e_i = (p_i, p_{i+1})$  for any  $i$  s.t.  $1 \leq i \leq n - 1$  contains the shortest path tree of  $v_i$  in  $H$ .*

**Proof:** Note that the weight of the above mentioned path, i.e.  $\pi = \{s = p_0, p_1, \dots, p_n, v_n\} \cup path_H(v_n, v_i)$ . where " $\cup$ " is the union operator, is given by  $\|\pi\| = d_H(v_n, v_i)$ . Now let us consider another path, say  $\pi_2$  from  $s$  to  $v_i$  which leaves  $P$  at some vertex  $p_a$  with  $a < n$ . That is,  $\pi_2 = \{s=p_0, p_1, \dots, p_a, v_a\} \cup path_H(v_a, v_i)$ . The weight of this

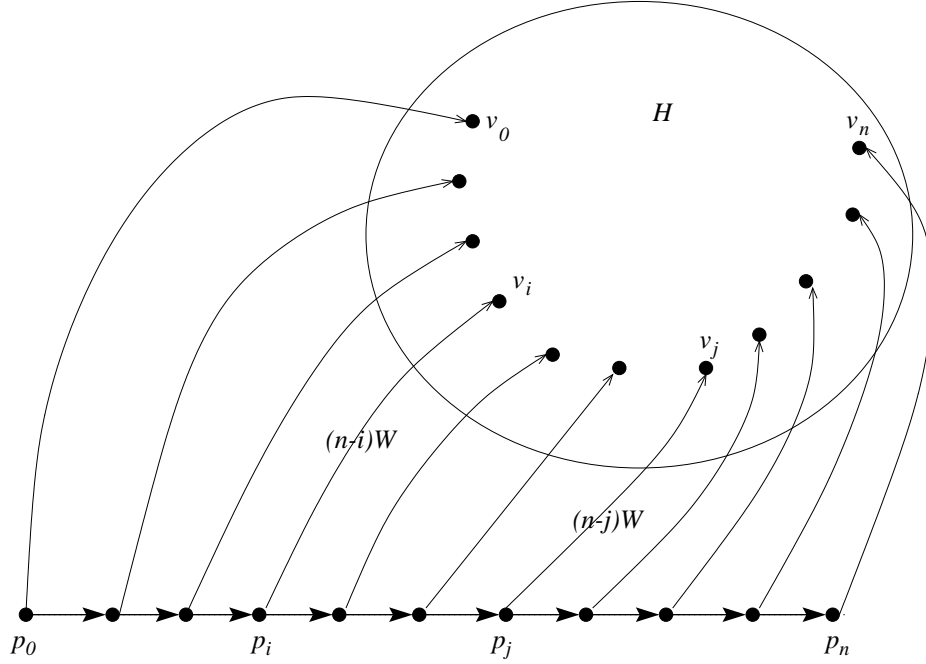


Figure 4.3: Construction for Shortest Paths Tree Maintenance Lower Bound

path satisfies :

$$\|\pi_2\| \geq (n - a)W \geq W > \|\pi\|$$

Recall that the choice of  $W$  makes it larger than any simple path in  $H$ . Thus, the shortest path from  $s$  to any node  $v \in H$  passes through  $v_n$ . It follows from the *subpath optimality* property of shortest paths that the subpath  $path(v_n, v)$  contained in  $path(s, v)$  is the shortest path from  $v_n$  to  $v$ . And hence, the shortest path tree of  $v_n$  in  $H$  is a subtree of the shortest path tree of  $s$  in  $G$ . The proof for the second part of the lemma follows from a similar argument. (Note that on deleting the edge  $e_i = (p_i, p_{i+1})$ , the vertices  $p_{i+1}, p_{i+2}, \dots, p_n$  become unreachable from  $s$ . But this does not affect the argument for the lower bound. Moreover, we can argue that computing the replacement shortest paths trees for the  $n$  edges of  $P$  provides the information about the all pair shortest paths computation in  $H$ , and computing the replacement trees for other edges of the original shortest paths tree of  $s$  is additional work.)  $\square$

Thus, computing the shortest paths trees in each of the  $n$  graphs obtained from  $G$  by

deleting exactly one edge on  $P$  generates the shortest paths trees of all the  $n$  nodes in  $H$ . Substituting  $H$  by the specific tripartite graph used by Karger-Koller-Phillips in their APSP lower bound construction we arrive at the lower bound of  $\Omega(mn)$  for the problem being considered. Note that augmenting the tripartite graph by  $P$  and the connecting edges does not alter the asymptotic size of the graph and  $G$  still has  $O(n)$  nodes and  $O(m)$  edges. More formally, we have the following theorem for the said problem :

**Theorem:** *Any path-comparison based algorithm for the replacement shortest paths tree for a directed graph on  $n$  nodes and  $m$  edges must spend at least  $\Omega(mn)$  time.*

**Proof:** We use the tripartite graph used by Karger-Koller-Phillips [KKP91] in their lower bound construction as the graph  $H$  in our construction. We use the same weights for the edges as used by them, namely those given by equations (3.1) and (3.2). Next, we form the graph  $G$  as described above. We argue that the algorithm needs to investigate all the  $\Theta(mn)$  triples in  $H$ . Else, if the algorithm ignores the triple  $(x_{a^*}, y_{b^*}, z_{c^*})$ , we modify the weights of some edges such that the shortest path tree of  $s$  reported for the graph  $G \setminus e_{a^*}$ , with  $e_{a^*} = (p_{a^*}, p_{a^*+1})$ , is incorrect, since after the modification, the shortest path to  $z_{c^*}$  passes through  $x_{a^*}$  and  $y_{b^*}$  which the algorithm cannot detect since all other paths maintain their relative order. We modify the edge weights as specified by equations (3.3) and (3.4). It is trivial to check that after modification,  $path_G(s = p_0, z_{c^*})$  passes through  $x_{a^*}$  and  $y_{b^*}$ . In order to verify that all the remaining paths maintain their relative order, let us consider any two paths  $\pi_1$  and  $\pi_2$  in  $G$ . If these paths have different *connecting* edges, then their relative order is dictated solely by the weight of these edges and remains unchanged since the difference in the weights of these edges is much larger than the change in their weights due to the modifications. In case they use the same connecting edges, even then their relative order remains intact, following the argument by Karger-Koller-Phillips in [KKP91]. This completes the proof.  $\square$

Interestingly, for the undirected version, a similar construction yields the lower bound

of  $\Omega(\text{All Pairs Shortest Paths})$ , but unfortunately there is no lower bound for the undirected version of the problem.

## 4.4 Replacement Paths for Subpaths Deletions

Another variation of the basic replacement paths problem deals with computing the replacement paths on deletions of subpaths of the given  $(s, t)$  shortest path in  $G$  as opposed to single edge deletions. Specifically, given a directed graph  $G$  with non-negatively weighted edges, two specified nodes  $s$  and  $t$ , a shortest path  $P$  from  $s$  to  $t$ , and a set of  $Q = \{p_1, p_2, \dots, p_k\}$  of  $k$  subpaths of  $P$ , we need to compute the shortest path from  $s$  to  $t$  in each of the  $k$  graphs obtained from  $G$  by deleting exactly one subpath in  $Q$ .

A naive approach towards solving this problem would require  $O(k(m + n \log n))$  using a single source shortest path computation for each of the deleted subpaths. Since  $k$  can be as large as  $O(n^2)$ , this bound can be  $O(mn^2 + n^3 \log n)$  in the worst case. We prove a lower bound of  $\Omega(m\sqrt{k})$  and an upper bound of  $O(mn + n^2 \log n + kn)$ . We consider some variations of this basic problem including one where the given subpaths are disjoint, and one where they overlap arbitrarily. The lower bound holds even in the case where the given subpaths overlap in a very “orderly” fashion forming a staircase like structure. If the subpaths are disjoint, the undirected version of the problem can be solved in optimal  $O(m + n \log n)$  time using a technique by Nardelli, Proietti and Widmayer [NPW01] and we present a brief sketch of the technique here.

### 4.4.1 Disjoint Subpaths

#### Replacement Paths for Node Deletions

We now list the key features of the algorithm by Nardelli, Proietti and Widmayer [NPW01] for the *node* version of the replacement paths problem and show how the technique can be used to efficiently solve the *disjoint subpaths deletions* version.

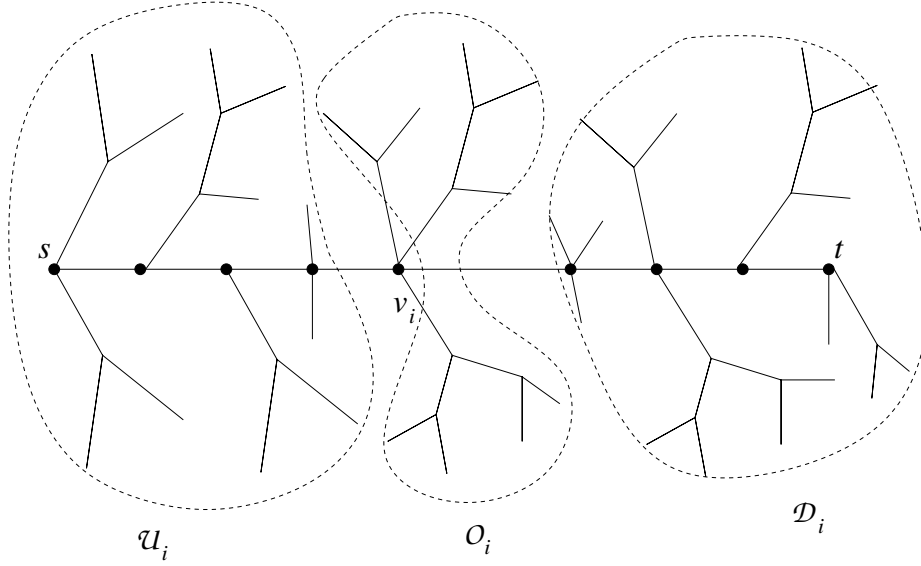


Figure 4.4: Node version of the replacement paths problem for undirected graphs

Let us consider the properties of the graph  $G \setminus v_i$  when we need to find the replacement path for the node  $v_i$ . See figure (4.4). On deleting the node  $v_i$ , the vertices of  $G$  are split into three sets.  $U_i$  is the set of vertices attached to  $s$  in  $T_s \setminus v_i$ ,  $D_i$  consists of those attached to  $t$  in  $T_t \setminus v_i$ , and finally, the remaining ones (except  $v_i$ ) form the set  $O_i$ . Here,  $T_s$  refers to the shortest paths tree of  $s$  in  $G$ . Note that the shortest path in  $G$  from  $s$  to any vertex  $v \in O_i$  passes through  $v_i$ . Following are some properties crucial to the design of the algorithm:

- $U_i \cup O_i \cup D_i = V \setminus \{v_i\}$
- $U_i, O_i$  and  $D_i$  are pairwise disjoint
- $U_i \subset U_{i+1}$
- $D_{i+1} \subset D_i$
- $O_i$  and  $O_j$  are disjoint

We state two important lemmas without proofs:

**Lemma 1** [NPW01] *For each node  $u \in U_i$ ,  $d_G(s, u) = d_{G \setminus v_i}(s, u)$ .*

**Lemma 2 [NPW01]** For each node  $u \in D_i$ ,  $d_G(u, t) = d_{G \setminus v_i}(u, t)$ .

It is easy to verify these lemmas and the reader is referred to [NPW01] for detailed proof of the second lemma. Using the lemmas and the properties listed above, the replacement path for the node  $v_i$  can be expressed as :

$$d_{G \setminus v_i}(s, t) = \text{MIN}_{(u,v) \in E(U_i \cup O_i, D_i)} \{d_{G \setminus v_i}(s, u) + \text{weight}(u, v) + d_{G \setminus v_i}(v, t)\}$$

This immediately suggest the remaining algorithm. Only the first term in the above expression remains to be computed for all  $u \in O_i$ . An easy modification of the Dijkstra's shortest path algorithm [Dij59] allows us to compute the  $s, u$  distances for all vertices  $u \in O_i$  in  $O(m_i + n_i \log n_i)$  time, where  $m_i = |E(U_i, O_i) \cup E(O - i, O_i)|$  and  $n_i = |O_i|$ . Since the  $O_i$ 's are disjoint, computing these values for all the  $O_i$ 's is

$$\sum_{i=1}^{k-1} O(m_i + n_i \log n_i) = O(m + n \log n)$$

where  $k$  is the number of intermediate nodes on the shortest path from  $s = v_0$  to  $t = v_k$ . The reader is referred to [NPW01] for the detailed analysis and the complete algorithm.

## Replacement Paths for Disjoint Subpaths Deletions

We now briefly show how the disjoint subpaths version and the node version are essentially the same. See figure (4.5).

On deleting the subpath  $p_i = \{v_j, v_{j+1}, \dots, v_l\}$  from the graph, the vertices are split into several components. We group them into three sets as was done in [NPW01]:  $U_i$  is the set containing vertices connected to  $s$  in  $T_s \setminus p_i$ ,  $D_i$  is the set of vertices connected to  $t$  in  $T_s \setminus p_i$  and finally, the remaining ones (except those in  $p_i$ ) form the set  $O_i$ . It is easy to verify that all the properties and lemmas mentioned above and as listed in [NPW01] continue to hold in this case as well. Proceeding in exactly the same way as in [NPW01], we arrive at the (near) optimal bound of  $O(m + n \log n)$  for this problem.

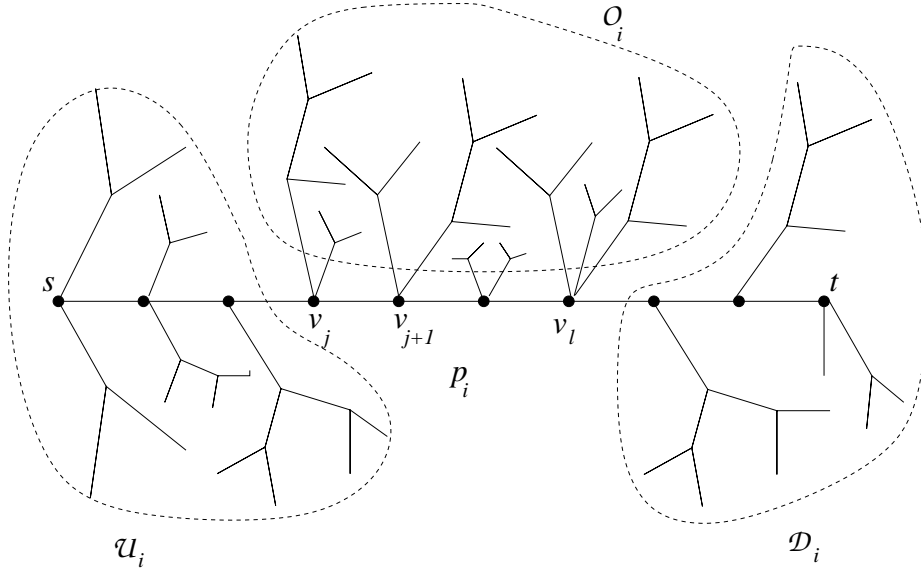


Figure 4.5: Disjoint subpaths deletions in undirected graphs

#### 4.4.2 Lower Bound

See figure (4.6) for a lower bound construction. Our graph  $G$  has a spine of  $2n$  nodes. That is,  $P = \{s = s_1, s_2, \dots, s_n, d_n, d_{n-1}, \dots, d_1 = t\}$ , one  $s_i$  for each of the  $n$  source nodes in the graph  $H$ , and one  $d_j$  for each of the  $n$  destination nodes in  $H$ . The weights of the edges of  $P$  are 0. The choice of  $W$  is basically the same as in other cases :  $W = 10nw$  where  $w$  is the heaviest edge in  $H$  (we have the factor of 10 here since the number of nodes in  $H$  is  $3n$ ).  $H$ , for our lower bound proof, is the specific tripartite graph used by Karger-Koller-Phillips in their APSP lower bound construction [KKP91]. The weights of the edges connecting the source and destination nodes in  $H$  to the  $s_i$  and  $d_j$  nodes respectively on  $P$  are given by :

$$\|s_i, v_i\| = (n - i)W \quad (4.1)$$

$$\|u_j, d_j\| = (n - j)W \quad (4.2)$$

Notice the slight variation in the assignment of weights in the equation (4.2). This happens due to the *reverse* numbering of the  $d$  nodes and the basic idea remains the

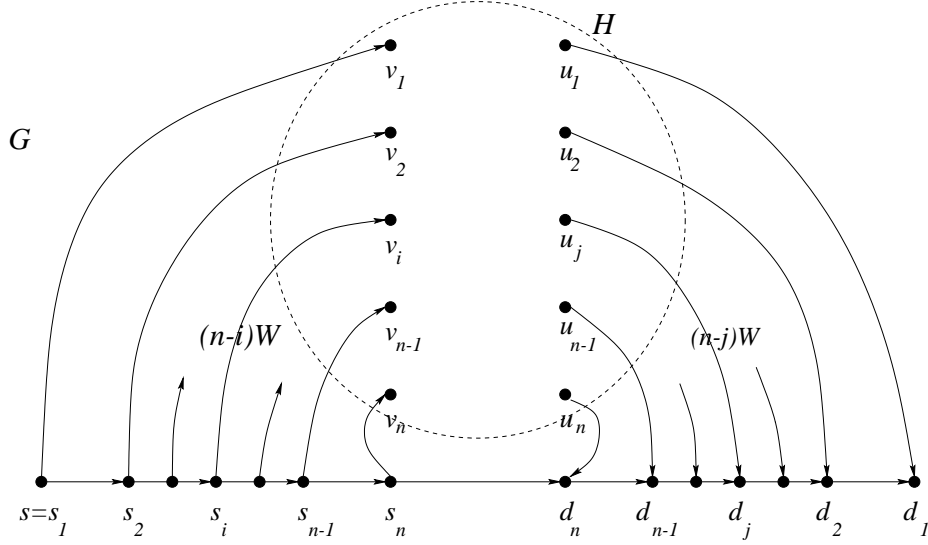


Figure 4.6: Construction for Subpaths Deletions Lower Bound

same, namely that the edges directed from the vertices of  $H$  to those of  $P$  should increase in weight as we move from left to right along  $P$ .

The specific assignment of weights to the connecting edges as specified in equations (4.1) and (4.2) yields the following lemma :

**Lemma:** *The replacement path on deleting the subpath  $(s_i, d_j)$  of  $P$  has weight exactly given by :*

$$\|path_{s,t;G \setminus path(s_i, d_j)}\| = (2n - i - j)W + d_H(v_i, u_j) \quad (4.3)$$

**Proof:** First, notice that the path  $p(s, t) = \{s = s_1, s_2, \dots, s_i, v_i\} \cup path_H(v_i, u_j) \cup \{d_j, d_{j-1}, \dots, d_1 = t\}$  has weight precisely equal to that given by equation (4.3). Let us now consider some other candidate replacement path. Any replacement path  $\pi$  that leaves  $P$  at some vertex  $s_a$  with  $a < i$  has weight satisfying  $\|\pi\| \geq (n - i + 1)W + (n - j)W = (2n - i - j)W + W > \|p(s, t)\|$  and is hence heavier than the path  $p(s, t)$  defined above since the choice of  $W$  makes it heavier than any simple path in  $H$  (note that any candidate path *cannot* join  $P$  before  $d_j$ ).

Similarly, any candidate path which joins  $P$  after  $d_j$  is heavier than  $p(s, t)$ . This completes the proof.  $\square$

Let us now consider the set of subpaths to be  $Q = \{(s_i, d_j)\}, \forall i, j \text{ s.t. } 1 \leq i, j \leq n$ . As established by the above lemma, the set of replacement paths for all the subpaths in  $Q$  provides us with enough information to derive the shortest path distances from all the source vertices in  $H$  to all the destination vertices (in  $H$ ). Using the lower bound proof by Karger-Koller-Phillips, we arrive at a lower bound of  $\Omega(mn)$  for computing the replacement paths for this set of  $k = |Q| = \Theta(n^2)$  subpaths.

Using the graph  $H$  as the one used in the lower bound construction of the  $K$ -pairs shortest paths problem, we can establish a bound of  $\Omega(m\sqrt{k})$  for this problem. In this construction, the indices of the  $s$  and  $d$  nodes on the spine  $P$  would vary from 1 to  $\sqrt{k}$  each and the set  $Q$  would be the set of all possible subpaths of  $P$  and  $k = |Q| = \Theta(K)$ .

We are now ready to state the following theorem for the results in this section :

**Theorem:** *Any path-comparison based algorithm for computing the replacement paths for a given set of  $k$  subpaths of the shortest path from a node  $s$  to a node  $t$  must spend at least  $\Omega(m\sqrt{k})$  time.*

**Proof:** We use the specific tripartite graph used in the  $k$ -pairs shortest paths lower bound construction as the graph  $H$  in our construction and form a graph  $G$  with  $O(n)$  nodes and  $O(m)$  edges by *connect* it to a spine  $P$  of  $2n$  edges as described above. The weights of the edges in  $H$  are assigned as specified in equations (3.1) and (3.2), while the *connecting* edges have weights specified by equations (4.1) and (4.2). We next specify a set  $Q$  of  $k$  subpaths of  $P$  and argue that in order to compute the shortest path from  $s$  to  $t$  in each of the  $k$  graphs  $G \setminus p_i, p_i \in Q$ , any (path-comparison based) algorithm must investigate all the  $\Theta(m\sqrt{k})$  triples. Else, if the algorithm ignores the triple  $(x_{a^*}, y_{b^*}, z_{c^*})$ , we can modify the edge weights as specified by equations (3.3) and (3.4) such that the algorithm errs in reporting the  $s - t$  distance in the graph  $G \setminus p_{a^*, c^*}$

where  $p_{a^*,c^*} = \{s_{a^*}, s_{a^*+1}, \dots, d_{c^*}\}$ . The modifications ensure that  $path_{G \setminus p_{a^*,c^*}}(s, t)$  passes through  $x_{a^*}, y_{b^*}$  and  $z_{c^*}$ . Also, all other paths maintain their relative order: let  $\pi_1$  and  $\pi_2$  are two paths being investigated. If they use different connecting edges, then their relative order is defined by these edges and remains preserved since the change in weights is much lesser than the difference in their weights. If they use the same connecting edge, then the argument by Karger-Koller-Phillips shows that the modification of edge weights does not alter their order. And, since the algorithm ignored this triple, and all other paths maintain their relative order, it reports the incorrect distance. This completes the proof.  $\square$

### 4.4.3 Upper Bound

Here we present the upper bound of  $O(APSP + kn)$  for the problem under consideration. In the path comparison model, this bound equals  $O(mn + n^2 \log n + kn)$ . For large value of  $k$ , say  $k = \Omega(n^2)$ , this becomes an  $\Theta(n^3)$  algorithm (and thus close to optimal owing to the lower bound of  $\Omega(mn)$ ). The basic technique is exactly same as the one described in the  $O(APSP)$  algorithm for the basic replacement paths problem for directed graphs.

We do an all pairs shortest paths computation on the graph  $G'$  obtained from the given graph  $G$  by deleting all the edges on the given shortest path  $P$  from  $s$  to  $t$ . We sort the given set of  $k$  subpaths in decreasing order of their *right* end points. Notice that deleting a subpath  $p_i$  of  $P$  partitions the graph  $G$  into several vertex sets. Denote the set containing  $s$  by  $V_s$ . Notice that the shortest path from  $s$  to any vertex  $v \in V_s$  remains intact. Furthermore, the shortest path from  $s$  to  $t$  in the graph  $G \setminus p_i$  has as its prefix, the shortest path from  $s$  to some vertex  $v \in V_s$ . If  $P = \{s = u_1, u_2, \dots, u_p = t\}$  and  $p_i = \{u_a, u_{a+1}, \dots, u_b\}$ , the best replacement path from  $s$  to  $t$  exiting  $V_s$  at a vertex  $v \in V_s$  is given by :

$$path_v(s, t) = MIN_{j=b}^p \{d_G(s, v) + d_{G'}(v, u_j) + d_G(u_j, t)\} \quad (4.4)$$

Also, the paths involved in the above minimization do not include the edges of  $p_i$ , owing to the construction of  $G'$  from  $G$ . Finally, the best replacement path for the

subpath  $p_i$  would be the minimum of the replacement paths through all the nodes in  $V_s$ . That is,

$$path_{G \setminus p_i}(s, t) = MIN_{v \in V_s} path_v(s, t) \quad (4.5)$$

To begin computing the replacement paths for the deleted subpaths, we start in an orderly fashion in accordance with the reverse sorted order of the right end points of the subpaths, the advantage being that this makes it faster and easier to update the  $path_v(s, t)$  values. Having computed the replacement path for the subpath  $p_{i+1} = \{u_a, \dots, u_b\}$ , when doing the same for  $p_i = \{u_c, \dots, u_d\}$ , we update the  $path_v(s, t)$  values in the following way :

$$path_v(s, t) = \min\{path_v(s, t), \min_{j=d}^{b-1}\{d_G(s, v) + d_{G'}(v, u_j) + d_G(u_j, t)\}\} \quad (4.6)$$

The analysis of the algorithm is straight forward :  $|V_s|$  is at most  $O(n)$ ; the execution of equation (4.6) takes time at most  $O(n + k)$  since at most  $n - 1$  times  $b \neq d$ , thus consuming a total time of  $O(n(n + k))$  during the entire execution; and finally, the execution of equation (4.5) consuming  $O(n)$  time per subpath in  $Q$ . Thus, the time required *after* the initial APSP computation and sorting is  $O(kn + n^2)$ . Including these two computations, we arrive at a time complexity of  $O(APSP + k \log k + kn + n^2) = O(APSP + kn)$ .

## Chapter 5

# Concluding Remarks and Open Problems

We have proved a non-trivial super-linear lower bound of  $\Omega(\min(n^2, m\sqrt{n}))$  on the replacement paths problem for directed graphs when no such bound was previously known. Also, we have shown the existence of a solution for the  $k$ -pairs shortest paths problem from a solution for the replacement paths problem, where the former is a natural generalization of the well-studied *single source shortest paths* and *all pair shortest paths* problems, with  $1 \leq k \leq n^2$ . We proved a lower bound of  $\Omega(\min(nk, m\sqrt{k}))$  for the same. The lower bound result of  $\Omega(mn)$  on the *replacement shortest paths trees* (somewhat surprisingly) shows that the naive algorithm is optimal for graphs with  $m = \Omega(n \log n)$  and near-optimal for sparse graphs.

It is interesting to note that the construction for the basic replacement paths lower bound involves cyclic graphs. This is not surprising especially since both the replacement paths and the  $k$ -shortest paths problems can be solved much more efficiently for DAGs. The undirected graph algorithm in [HS01] works for DAGs and solves the replacement paths problem in  $O(m + n \log n)$  time while the  $k$ -shortest paths problem has been solved in  $O(m + n \log n + k)$  time [Epp94] using a technique not based on replacement paths.

Although our results hold only for directed graphs, the constructions for the cases of subpaths deletions and replacement shortest paths trees imply a solution for the all pair shortest paths problems for undirected graphs and we conjecture that the corresponding lower bounds would hold even in the undirected case (if not, an improvement is implied for the all pair shortest paths problem for the undirected graphs).

The upper bound of  $O(\text{All pairs shortest paths})$  shows that an improvement in the all pair shortest paths problem would imply an improvement for the replacement paths. Obviously, such an algorithm cannot be path-comparison based and would have to use techniques like fast matrix multiplication as in [Fre76, Tak92, Tak95, Zwi98]. As discussed earlier, all the present algorithms for the  $k$ -shortest paths use the replacement paths as a subroutine [Law72, HSB01, KIM82], yielding a time complexity of  $O(k.APSP)$ .

An obvious open problem suggested by our work is to bridge the gap between the lower bound and the naive upper bound for the replacement paths problem and the  $k$ -pairs shortest paths problem. With regard to the  $k$ -shortest paths problem, is it possible to solve the problem without using the replacement paths as a subroutine, as was done for the non-simple paths problem [Epp94] ? Can the weakness of the path comparison model for shortest path algorithms [HSB02] be exploited to design faster algorithms, even if just for theoretical interest ? Walking away from the path comparison model, can something better be done using techniques like fast matrix multiplication for the replacement paths or the  $k$ -pairs shortest paths problems, as has been done for the traditional shortest paths problems [Tak92, Tak95, Zwi98, SZ99, GM97a, GM97b] ?

# Bibliography

- [AT02] A. Archer and E. Tardos. Frugal path mechanisms. In *Proc. 13th Annu. ACM-SIAM Sym. Disc. Algorithms*, pages 991-999, 2002.
- [Bel58] R. Bellman. On a routing problem. In *Quart. Appl. Math.*, volume 16, pages 87 - 90, 1958.
- [BGV89] M.O. Ball, B.L. Golden, and R.V. Vohra. Finding the most vital arcs in a network. In *Oper. Res. Letters*, pages 8:73-76, 1989.
- [BKS95] A. BarNoy, S. Khuller, and B. Schieber. The complexity of finding most vital arcs and nodes. Technical Report CS-TR-3539, University of Maryland, Institute for Advanced Computer Studies, MD. 1995.
- [BS95] A. Brander and M. Sinclair. A comparative study of  $K$ -shortest paths algorithms. In *Proc. of 11th UK Performance Engineering Workshop*, pages 370-379, 1995.
- [Cho] R.A. Chowdhury. Improved distance oracles for avoiding link-failure. unpublished manuscript.
- [CSC02] H. Choi, S. Subramaniam, and H. Choi. On double-link failure recovery in WDM optical networks. In *IEEE INFOCOM*, 2002.
- [DI01] C. Demetrescu and G. F. Italiano. Fully dynamic all pairs shortest paths with real edge weights. In *IEEE Symposium on Foundations of Computer Science*, pages 260-267, 2001.

- [Dij59] E.W. Dijkstra. A note on two problems in connection with graphs. In *Numerische Mathematik*, pages 1:269-271, 1959.
- [DT02] C. Demetrescu and M. Thorup. Oracles for distances avoiding a link-failure. In *13th IEEE Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 838-843, 2002.
- [EG85] S. Even and H. Gazit. Updating distances in dynamic graphs. In *Methods of Operations Research*, pages 49:371-387, 1985.
- [Epp94] D. Eppstein. Finding the  $k$  shortest paths. In *IEEE Symposium on Foundations of Computer Science*, pages 154-165, 1994.
- [FF62] L.R. Ford and D.R. Fulkerson. Flows in networks. In *Princeton Univ. Press, Princeton, NJ*, 1962.
- [Flo62] R.N. Floyd. Algorithm 97, shortest path. In *Comm. ACM*, page 5:345, 1962.
- [FMSN98] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Semi-dynamic algorithms for maintaining single-source shortest path trees. *Algorithmica*, 22(3):250-274, 1998.
- [FMSN00] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Fully dynamic algorithms for maintaining shortest paths trees. *Journal of Algorithms*, pages 34:351-381, 2000.
- [Fre76] M.L. Fredman. New bounds on the complexity of the shortest path problem. In *SIAM J. of Computing*, pages 5:83-89, 1976.
- [FT87] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596-615, 1987.
- [FT00] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *INFOCOM (2)*, pages 519-528, 2000.

- [GM97a] Z. Galil and O. Margalit. All pairs shortest distances for graphs with small integer length edges. In *Information and Computation*, pages 134(2):103-139, 1997.
- [GM97b] Z. Galil and O. Margalit. All pairs shortest paths for graphs with small integer length edges. In *Journal of Computer and System Sciences*, pages 54(2):243-254, 1997.
- [HC99] E. Hadjiconstantinou and N. Christofides. An efficient implementation of an algorithm for finding  $K$  shortest simple paths. In *Networks*, pages 34(2):88-101, 1999.
- [HS01] J. Hershberger and S. Suri. Vickrey prices and shortest paths: What is an edge worth? In *IEEE Symposium on Foundations of Computer Science*, pages 252-259, 2001.
- [HSB01] J. Hershberger, S. Suri, and A. Bhosle. Finding the  $K$  shortest simple paths. Technical Report, University of California, Santa Barbara. 2001.
- [HSB02] J. Hershberger, S. Suri, and A. Bhosle. On the difficulty of some shortest paths problems (Submitted for review). 2002.
- [KIM82] N. Katoh, T. Ibaraki, and H. Mine. An efficient algorithm for  $k$  shortest simple paths. In *Networks.*, pages 12:411-427, 1982.
- [Kin99] V. King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *IEEE Symposium on Foundations of Computer Science*, pages 81-91, 1999.
- [KKP91] D. R. Karger, D. Koller, and S. J. Phillips. Finding the hidden path: Time bounds for all-pairs shortest paths. In *IEEE Symposium on Foundations of Computer Science*, pages 560-568, 1991.
- [KR84] D. Kirkpatrick and S. Reisch. Upper bounds for sorting integers on random access machines. In *Theoretical Computer Science*, pages 28(3):263-276, 1984.

- [KS99] V. King and G. Sagert. A fully dynamic algorithm for maintaining the transitive closure. pages 492-498, 1999.
- [Law72] E.L. Lawler. A procedure for computing the  $k$  best solutions to discrete optimization problems and its application to the shortest path problem. In *Management Science.*, pages 401-405, 1972.
- [MFB99] M. Medard, S.G. Finn, and R.A. Barry. WDM loop-back recovery in mesh networks. In *IEEE INFOCOM*, pages 752-759, 1999.
- [MMG89] K. Malik, A.K. Mittal, and S.K. Gupta. The  $k$  most vital arcs in the shortest path problem. In *Oper. Res. Letters*, pages 8:223-227, 1989.
- [MP00] E. Martins and M. Pascoal. A new implementation of Yen's ranking loopless paths algorithm. Submitted for publication, Universidade de Coimbra, Portugal. 2000.
- [MPS97] E. Martins, M. Pascoal, and J. Santos. A new algorithm for ranking loopless paths. Technical Report, Universidade de Coimbra, Portugal. 1997.
- [NPW01] E. Nardelli, G. Proietti, and P. Widmayer. Finding the most vital node of a shortest path. In *COCOON*, pages 278-287, 2001.
- [NR99] N. Nisan and A. Ronen. Algorithmic mechanism design. In *Proc. 31st Annu. ACM Sym. Theory of Comput.*, pages 129-140, 1999.
- [Per86] A. Perko. Implementation of algorithms for  $K$  shortest loopless paths. In *Networks*, pages 16:149-160, 1986.
- [PS80] W.J. Paul and J. Simon. Decision trees and random access machines. In *Proc. 31st Annu. ACM Sympos. Theory Comput.*, pages 331-340, 1980.
- [Roh85] H. Rohnert. A dynamization of the all-pairs least cost problem. In *Proc. 2nd Annual Symposium on Theoretical Aspects of Computer Science*, pages 279-286, 1985.

- [Sei95] R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. In *Journal of Computer and System Sciences*, pages 51(3):400-403, 1995.
- [Spi73] P.M. Spira. A new algorithm for finding all shortest paths in a graph of positive arcs in average time  $O(n^2 \log^2 n)$ . In *SIAM J. Comput.*, pages 2:28-32, 1973.
- [SZ99] A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *IEEE Symposium on Foundations of Computer Science*, pages 605-615, 1999.
- [Tak92] T. Takaoka. A new upperbound on the complexity of the all pairs shortest path problem. In *Information Processing Letters* 43, pages 195-199, 1992.
- [Tak95] T. Takaoka. Sub-cubic cost algorithms for the all pairs shortest path problem. In *Workshop on Graph-Theoretic Concepts in Computer Science*, pages 323-343, 1995.
- [Yen71] J.Y. Yen. Finding the  $k$  shortest loopless paths in a network. In *Management Science*, pages 17:712-716, 1971.
- [Yen72] J.Y. Yen. Another algorithm for finding the  $k$  shortest loopless network paths. In *Proc. of 41st Mtg. Operations Research Society of America.*, volume 20, 1972.
- [Zwi98] U. Zwick. All pairs shortest paths in weighted directed graphs exact and almost exact algorithms. In *IEEE Symposium on Foundations of Computer Science*, pages 310-319, 1998.