On the Difficulty of Some Shortest Path Problems

John Hershberger* Subhash Suri† Amit Bhosle‡

Abstract

We prove super-linear lower bounds for some shortest path problems in directed graphs, where no such bounds were previously known. 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 $P = \{e_1, e_2, \ldots, e_p\}$ 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 one of the edges e_i. We show that the replacement paths problem requires $\Omega(m\sqrt{n})$ time in the worst case whenever $m = O(n\sqrt{n})$. This also establishes a similar lower bound for computing the *second shortest* simple path by any algorithm that uses replacement paths as a subroutine; all known algorithms for the k shortest paths fit this description. To put our lower bound in perspective, we note that both these problems (replacement paths and second shortest path) can be solved in near linear time for *undirected* graphs. We also give a nearly tight lower bound of $\Omega(nm)$ for computing the shortest path distances between a fixed source s and all other nodes if each edge of the shortest path tree is removed in turn.

1 Introduction

Some shortest path problems seem to have resisted efficient algorithms for *directed* graphs, while their *undirected* counterparts have been solved to optimality or near optimality. One notorious example is the problem of determining the k shortest simple paths between a pair of nodes. Specifically, given a directed graph G with non-negative edge weights, a positive integer k, and two vertices s and t, the problem asks for the k shortest paths from s to t in increasing order of length, where the paths are required to be *simple* (loop free); if the paths are allowed to be *non-simple*, then an optimal algorithm is known for both directed and undirected graphs [7]. The best algorithm known for computing the k shortest simple paths in a directed graph is due to Yen [24, 25]. The worst-case time complexity of his algorithm, using modern data structures, is $O(kn(m + n\log n))$; in other words, it requires $\Theta(n)$ single-source shortest path computations for each of the k shortest paths that are produced. In the past thirty years, there have been several "practical" improvements to Yen's algorithm, but none has succeeded in improving the worst-case asymptotic complexity of the problem. (See the references [6, 10, 18, 19, 23] for implementations and modifications of Yen's algorithm.) By contrast, if the graph G is undirected, then one can compute the k shortest simple paths in time $O(k(m + n\log n))$ [12, 14].

A second problem with similar behavior is the *replacement paths* problem. Given a directed graph G with non-negative edge weights, and a pair of nodes s, t, let $P = (e_1, e_2, \ldots, e_p)$ denote

*Mentor Graphics Corp., 8005 SW Boeckman Road, Wilsonville, OR 97070. Email: john_hershberger@mentor.com

†Department of Computer Science, University of California, Santa Barbara, CA 93106. Email: suri@cs.ucsb.edu.
Supported in part by National Science Foundation grants IIS-0121562 and CCR-9901958.

‡Department of Computer Science, University of California, Santa Barbara, CA 93106. Email: bhosle@cs.ucsb.edu.
Supported in part by National Science Foundation grants IIS-0121562 and CCR-9901958.

the sequence of edges in a shortest path from s to t in G. The replacement paths problem is to compute the shortest path from s to t, in each of the graphs $G \setminus e_i$, that is, G with edge e_i removed, for $i = 1, 2, \ldots, p$. (Another variant of the replacement paths problem arises when *nodes* of the path P are removed one at a time. These two problems have equivalent computational complexity.) The replacement paths problem is motivated both by routing applications, in which a new route is needed in response to individual link failures [8], and by computational mechanism design, in which the Vickrey-Clarke-Grove scheme is used to elicit true link costs in a distributed but self-interested communication setting, such as the Internet [3, 21]. 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 distance from s to t in the graph G with the cost of e_i set to zero. Computing all these payments is equivalent to solving the replacement paths problem for s, t.

A naïve algorithm for the replacement paths problem runs in $O(n(m+n \log n))$ time, executing the single-source shortest path algorithm up to n times, but no better algorithm is known. By contrast, if G is *undirected*, then the problem can be solved in $O(m + n \log n)$ time [11]; that is, roughly one single-source shortest path computation suffices! (Our FOCS 2001 paper [11] erroneously claims to work for both directed and undirected graphs; however, there is a flaw that invalidates the algorithm for directed graphs.) In the case of undirected graphs, the same bound is also achieved by Nardelli, Proietti, and Widmayer [20], who solve the *most vital node* problem with an algorithm that also solves the replacement paths problem. The work of Nardelli, Proietti, and Widmayer is in turn based on earlier work by Malik, Mittal and Gupta [17], Ball, Golden, and Vohra [4] and Bar-Noy, Khuller, and Schieber [5]. All of these algorithms are restricted to undirected graphs, and again the directed version of the problem seems to have eluded efficient solutions. One interesting connection between the two problems mentioned above (k simple shortest paths and replacement paths) is that all the known k shortest paths algorithms, including those for undirected graphs, solve the replacement paths problem as a subproblem.

Main results

Our main result is to prove a *lower bound* on the replacement paths problem in directed graphs, thereby explaining our lack of success in finding an efficient algorithm. Our lower bound applies to the *path comparison model* proposed by Karger, Koller, and Phillips [13]. In this model, an algorithm learns about shortest paths by comparing the lengths of two different paths. A path-comparison based algorithm can perform all the standard operations in unit time, but its access to edge weights is only through the comparison of two paths. Most of the known shortest path algorithms, including those of Dijkstra, Bellman-Ford, Floyd, Spira, Frieze-Grimmet, as well as the hidden paths algorithm of Karger-Koller-Phillips [13] fit into the path comparison model. On the other hand, the $o(n^3)$ time all-pairs shortest paths algorithms of Fredman [9] does not fit the path comparison model—it compares sums of weights of edges that do not necessarily form paths. The matrix-multiplication based algorithms by Alon, Galil, Margalit [1] and Zwick [26, 27] also fall outside the path comparison model; however, these algorithm assume the weights lie in a small integer range.

Despite its obvious appeal and power, we discovered that the path comparison model also has an unfortunate weakness: adding extra edges to a graph, even ones that obviously do not affect the solution to the problem being solved, can invalidate lower bounds. We discuss this limitation in Section 3.2. Because of this limitation, we must restrict our lower bound to algorithms that do not compare certain paths that cannot belong to any solution. All presently known algorithms for the replacement paths problem satisfy this restriction.

We show that the replacement paths problem requires $\Omega(m\sqrt{n})$ time in the worst case for a

directed graph G that has n vertices and m edges, with $m = O(n\sqrt{n})$. (Another way of phrasing the limitation on m is to say that the lower bound is $\Omega(\min(n^2, m\sqrt{n}))$.) Any k shortest paths algorithm that computes replacement paths as a subroutine, therefore, is also subject to this lower bound. In particular, even computing the *second shortest* simple path in a directed graph with $m = O(n\sqrt{n})$ edges requires $\Omega(m\sqrt{n})$ time. The same lower bound also applies to the replacement paths problem where intermediate nodes of the s–t path are removed one at a time. To the best of our knowledge, this is the first nontrivial lower bound for these problems. By contrast, near-linear time algorithms are known for these problems in undirected graphs.

Our main result yields similar lower bounds for a couple of other related problems. For instance, what is the complexity of the replacement paths problem if the metric of the path is $\langle\text{length}\rangle \times \langle\text{hop count}\rangle$? This metric has been suggested by Archer and Tardos [2, 3] as part of their work on frugal path mechanism design. Their motivation is that paths with many hops tend to require large payments to individual links, and so favoring paths with fewer hops might work better in practice. This leads to the same computational problem as the replacement paths problem, but using this more complex metric. We show that the $\Omega(m\sqrt{n})$ lower bound holds for this variant of the replacement paths problem as well.

We also establish a nearly tight lower bound of $\Omega(nm)$ for the edge-deletion problem in *shortest path trees*. Specifically, given the shortest path tree $T(s)$, rooted at a node s, the problem is to compute distances from s to all other nodes in each of the $n-1$ graphs $G \setminus e_i$, where $e_i \in T(s)$. Our $\Omega(nm)$ lower bound nearly matches the $O(n(m + n\log n))$ time upper bound one can get by n invocations of the single-source shortest path problem.

2 A Lower Bound Construction

Our main theorem establishes a lower bound on the replacement paths problem by tying the computation of replacement paths to computing shortest path distances between pairs of vertices in another graph. We define the following natural problem:

> ***k*-Pairs Shortest Paths Problem:** Given a directed graph G, with non-negative edge weights, and k source-destination pairs (s_i, t_i), for $i = 1, 2, \ldots, k$, compute the shortest path distance for each pair (s_i, t_i). (The sources and destinations need not be unique.)

Our lower bound relates the n pair distances in an instance of the n-pairs shortest paths problem to the replacement path distances in a "wrapper" graph. We describe this general construction below.

We start with a directed path $P = (v_0, v_1, \ldots, v_n)$, with $\|v_i, v_{i+1}\| = 0$, for $i = 0, 1, \ldots, n-1$, where $\|v_i, v_{i+1}\|$ is the length (weight) of the edge (v_i, v_{i+1}). Set $s = v_0$ and $t = v_n$ as the source and destination pair for our replacement paths problem. Let H be an arbitrary directed graph with n nodes, m edges, and non-negative edge weights. We choose n source-destination pairs (s_i, t_i) in H. Let w be a weight at least as large as the maximum weight of any simple path in H; we can crudely bound w by n times the largest edge weight in H. We choose $W = 10w$. We create edges between the nodes of the path P and the graph H as follows.

We map the n edges of P bijectively onto the n source-destination pairs (s_j, t_j) in H. Suppose the edge $(v_i, v_{i+1}) \in P$ is mapped to the pair (s_j, t_j). We create a directed edge (v_i, s_j) with

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

Similarly, we create a directed edge (t_j, v_{i+1}) with

$$\|t_j, v_{i+1}\| = (i + 1)W.$$

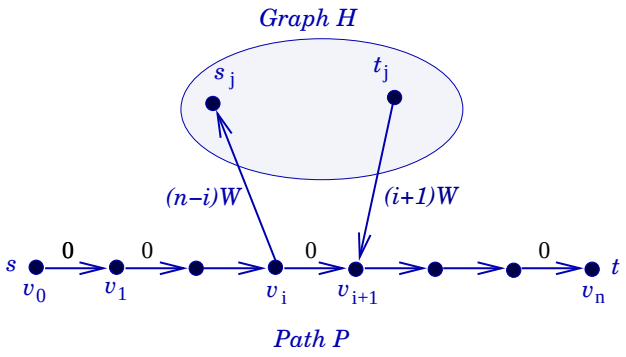

Figure 1: The main lower bound construction.

Figure 1 illustrates the construction. Note that the index of the pair (s_j, t_j) plays no rôle in the cost of these edges, but only influences which nodes of the path P the edges are joined to. We call the entire graph G; it has $O(n)$ vertices and $O(m)$ edges. Observe that P is the shortest path from s to t in G. The following lemma states a crucial property of this graph.

Lemma 2.1 *Suppose the edge $(v_i, v_{i+1}) \in P$ is mapped to the source-destination pair (s_j, t_j) in H. Then the shortest path from s to t in the graph $G \setminus (v_i, v_{i+1})$ has cost*

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

where $d_H(s_j, t_j)$ denotes the shortest path distance between s_j and t_j in the subgraph H.

Proof: We first note that the replacement path that starts at s, follows P up to node v_i, uses edge (v_i, s_j), takes the shortest path from s_j to t_j in H, returns to v_{i+1} using the edge (t_j, v_{i+1}), and finally traces the path P from v_{i+1} to t has length

$$(n-i)W + d_H(s_j, t_j) + (i+1)W \;=\; (n+1)W + d_H(s_j, t_j).$$

We next show that any other replacement path candidate is longer. Since the cost of edges connecting a vertex in P to a vertex of H is at least W, which is substantially larger than any subpath in P or any path in H, an optimal replacement path makes exactly one trip to H. Let us consider a path that follows P up to a node v_a, for $a \leq i$, and then returns to a node v_b, for $b \geq i+1$, where either $a \neq i$ or $b \neq i+1$. Supposing $a \neq i$, then this path has length at least $(n-a)W + (i+1)W \;\geq\; (n+2)W$. Since $W = 10w$, it follows that

$$(n+2)W \;>\; (n+1)W + d_H(s_j, t_j).$$

Similarly, any path for which $b \neq i+1$ is also longer. This completes the proof. ∎

In order to apply this construction to the replacement paths problem, we need a lower bound on the k-pairs shortest paths problem. While such a lower bound is easily derived by modifying the construction of Karger, Koller, and Phillips [13], a subtle point about the *robustness* of path comparison lower bounds must be made.

Definition 2.2 *A path comparison model lower bound for a graph problem is* path addition insensitive *if it holds even if the graph is modified by adding edges and vertices that do not affect the solution to the problem.*

Because Lemma 2.1 shows that computing the replacement paths for the graph G also computes the n-pairs shortest path distances for H, we have the following black-box reduction theorem.

Theorem 2.3 *If there is a directed graph H with n nodes, m edges, and n source-destination pairs (s_i, t_i) such that any algorithm to compute the shortest path distances between all the (s_i, t_i) pairs must take $f(n, m)$ time, and this lower bound is path addition insensitive, then there is a lower bound of $\Omega(f(n, m))$ time for the replacement paths problem on a graph with $O(n)$ nodes and $O(m)$ edges.*

3 The Path Comparison Model and Lower Bounds

We begin with a lower bound on the k-pairs shortest paths problem. Our argument uses a small modification of the construction of Karger, Koller, and Phillips [13]. Unfortunately, we then show in Section 3.2 that the Karger-Koller-Phillips lower bound is not path addition insensitive. As a result, the class of algorithms to which our lower bound result applies is slightly restricted, though it still includes all known algorithms for the replacement paths problem.

3.1 A lower bound for the k-pairs shortest paths problem

Our k-pairs shortest paths lower bound is based on the lower bound construction of Karger, Koller, and Phillips [13]. The graph is a directed tripartite graph, with vertices x_a, y_b, and z_c, where the

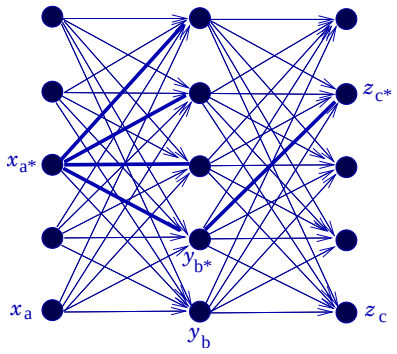

Figure 2: The lower bound construction of Karger, Koller and Phillips [13]. The edges whose weights are modified are shown as thick lines.

indices a, b, c range from 0 to $n-1$. The edges are of the form (x_a, y_b) or (y_b, z_c). The construction of [13] is for the all-pairs shortest paths problem, where the goal is to compute the shortest path distances $d(x_a, z_c)$, for all $0 \leq a, c \leq n-1$. See Figure 2 for an illustration. The lower bound argument depends on the fact that there is a choice of edge weights so that if the algorithm does not consider some x–z path, say $(x_{a^*}, y_{b^*}, z_{c^*})$, then an adversary can modify the weights in such a way that

1. The path $(x_{a^*}, y_{b^*}, z_{c^*})$ becomes the shortest path between x_{a^*} and z_{c^*}, and
2. The relative order of all the remaining paths remains unchanged.

Thus an algorithm is forced to consider all $\Theta(n^3)$ paths from x-nodes to z-nodes, giving a lower bound of $\Omega(n^3)$ path comparisons for the special case of $m = \Theta(n^2)$. In order to extend this construction to any value of m, we simply reduce the number of vertices in the middle column to

m/n, which gives $\Theta(mn)$ triples of the type (x_a, y_b, z_c). The specific weights used by Karger, Koller, and Phillips have the following form:

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

$$\|y_b, z_c\| = [0,\ 1,\ 0,\ c,\ 0,\ -b,\ 0]_{n+1} \quad (2)$$

using numbers in base $n+1$. That is, $[\phi_r, \ldots, \phi_0]_\alpha$ is defined as $[\phi_r, \ldots, \phi_0]_\alpha = \sum_{i=0}^{r} \phi_i \alpha^i$. The negative numbers do not cause any problems because the graph is a directed acyclic graph; they are introduced only to simplify the presentation. In order to make $(x_{a^*}, y_{b^*}, z_{c^*})$ the new shortest path, the weights of all the edges (x_{a^*}, y_b), for $b \leq b^*$, and the edge (y_{b^*}, z_{c^*}) are reduced as follows:

$$\|x_{a^*}, y_b\|' = [1,\ 0,\ a^*,\ 0,\ 0,\ b,\ b]_{n+1},\ \text{for all } b \leq b^* \quad (3)$$

$$\|y_{b^*}, z_{c^*}\|' = [0,\ 1,\ 0,\ c^*,\ 0,\ -b^*,\ -n]_{n+1} \quad (4)$$

It is now an easy exercise to check that using the new weights, the path $(x_{a^*}, y_{b^*}, z_{c^*})$ becomes the shortest path between x_{a^*} and z_{c^*}, while all other pairs of paths retain their relative ordering.

An easy modification of this construction leads to an $\Omega(\min(nk, m\sqrt{k}))$ lower bound for the k-pairs shortest paths problem. We use $\sqrt{k}$ vertices in the first and the third column, and $\min(n, m/\sqrt{k})$ vertices in the middle column. That is, for vertices x_a and z_c, the indices range from 0 to $\lceil\sqrt{k}\rceil$, and, for the y_b vertices, the indices range from 0 to $\min(n, \lceil m/\sqrt{k}\rceil)$. Thus, as long as $m = O(n\sqrt{k})$, then the total number of vertices in the graph is $O(n)$, the number of edges is $\Theta(m)$, and there are $\Omega(m\sqrt{k})$ triples of the form (x_a, y_b, z_c). A path-comparison based algorithm must examine all the triples to compute all x_a, z_c distances correctly.

Theorem 3.1 *For any n, k, and m, with $1 \leq k \leq n^2$ and $m = O(n\sqrt{k})$, there exists a directed graph H with n nodes, m non-negatively weighted 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).*

3.2 Limitations of path-comparison lower bounds

The path comparison lower bound of Karger et al. [13] is the only non-trivial lower bound known for the all-pair shortest paths problem. Since most shortest path algorithms fit into the path comparison model, such a lower bound has considerable intellectual value. Unfortunately, the lower bound is weaker than it appears at first glance. In particular, adding extra nodes and edges to the graph, even ones that clearly have no effect on the shortest paths under investigation, may invalidate the lower bound argument. The argument in [13] rests on the fact that any three-vertex path $(x_{a^*}, y_{b^*}, z_{c^*})$ can be made the shortest path linking x_{a^*} to z_{c^*} without affecting the relative order of the other paths. We show below that this crucial property is invalidated by the addition of some superfluous nodes and edges, which otherwise have no effect on the shortest path problem.

Consider the strongly-connected graph H' obtained from the directed acyclic graph H of Figure 2 by connecting the third column of vertices back to the first column by two fans of zero-weight edges and a single looping-back edge with weight W, where W is ten times the weight of the heaviest path in H. See Figure 3. Then for any pair (x_a, z_c), the shortest path joining them is identical to the shortest path in the original graph H. Furthermore, it is far from obvious how the existence of the additional edges in H' might make it easier to find the shortest path for any (x_a, z_c). Nevertheless, the additional edges invalidate Karger, Koller, and Phillips's lower bound by creating paths whose

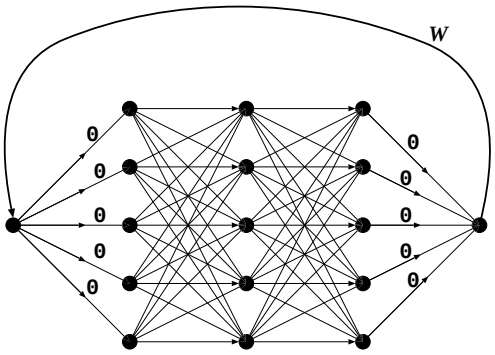

Figure 3: The lower bound breaks down when we add a useless loop.

relative order *does* change when the graph is re-weighted to make $(x_{a^*}, y_{b^*}, z_{c^*})$ the shortest path from x_{a^*} to z_{c^*}. For example, suppose $3 < b^* < n-2$ and $a^* \neq 0$. Consider the following two paths:

$$\begin{aligned} \pi_1 &= (x_{a^*}, y_3, z_0, \ldots \text{loop} \ldots, x_0, y_{n-2}, z_1) \\ \pi_2 &= (x_{a^*}, y_0, z_0, \ldots \text{loop} \ldots, x_0, y_{n-1}, z_1) \end{aligned}$$

In the unmodified graph, we have $\|\pi_1\| > \|\pi_2\|$, because the path weights are

$$\begin{aligned} \|\pi_1\| &= [2,\ 2,\ a^*,\ 1,\ n+1,\ -(n+1),\ 0]_{n+1} + W \\ \|\pi_2\| &= [2,\ 2,\ a^*,\ 1,\ n-1,\ -(n-1),\ 0]_{n+1} + W. \end{aligned}$$

After the edge weights are modified to make $(x_{a^*}, y_{b^*}, z_{c^*})$ the shortest path from x_{a^*} to z_{c^*}, the two path weights become

$$\begin{aligned} \|\pi_1\| &= [2,\ 2,\ a^*,\ 1,\ n-2,\ -(n-2),\ 3]_{n+1} + W \\ \|\pi_2\| &= [2,\ 2,\ a^*,\ 1,\ n-1,\ -(n-1),\ 0]_{n+1} + W, \end{aligned}$$

and now we have $\|\pi_1\| < \|\pi_2\|$. Thus, the relative order of $\|\pi_1\|$ and $\|\pi_2\|$ changes when the weights are modified, even though neither π_1 nor π_2 contains either edge (x_{a^*}, y_{b^*}) or (y_{b^*}, z_{c^*}). Thus, the lower bound of Karger, Koller, and Phillips is not path addition insensitive.

Another potential weakness of the path comparison model is that it allows *any* two paths to be compared at unit cost, without regard to the number of edges in the paths; that is, the computation is assumed to be free. But just as the unbounded word-size RAM model can be (theoretically) abused to sort numbers in $O(n)$ time by arithmetic on long numbers [15, 22], path comparison lower bounds can be invalidated by the creation of long paths, which may encode all relevant paths as subpaths. For instance, in our modification of the Karger-Koller-Phillips construction, we can create a single (non-simple) path that includes all the edges of the original graph, by using the loop over and over again. Once the algorithm is allowed to make comparisons among such long paths, it becomes difficult to argue about any unexamined subpath. In response to these shortcomings of the path comparison model, we limit the scope of our lower bound somewhat in the following section.

4 A Lower Bound on the Replacement Paths Problem

We now return to the replacement paths lower bound. We will use the construction described above, with $k = n$, as the subgraph H. If the Karger-Koller-Phillips construction were path addition

insensitive, this would immediately give an $\Omega(m\sqrt{n})$ lower bound on the replacement paths problem, by Theorem 2.3. However, because the k-pairs lower bound is not path addition insensitive, we must show that when the weights are modified to make a particular triple $(x_{a^*}, y_{b^*}, z_{c^*})$ the shortest path between x_{a^*} and z_{c^*}, then all relevant path orders are preserved. Since the graph H is supplemented with the path P and the edges connecting P and H, many new paths are introduced in the graph G whose orders are not considered in the lower bound argument of Section 3.1. In fact, if paths visit H more than once, then path orders may *not* be preserved, as in the example of Section 3.2.

Definition 4.1 *Given a path π from s to t, a* detour *is any subpath of π that begins and ends on P but contains no edges of P.*

We focus on algorithms that examine paths with at most one detour. This is reasonable, because of the following fact.

Lemma 4.2 *Given a directed graph G, and an instance of the replacement paths problem with s–t shortest path $P = (e_1, e_2, \ldots, e_p)$, suppose P_i is the replacement path when e_i is deleted, for $i = 1, 2, \ldots, p$. Then, P_i cannot have more than one detour.*

Proof: A detour that leaves P at some node x and returns to P on the subpath between s and x creates a cycle and can be shortened. (Recall that edges of G have non-negative costs.) If a path has two forward-going detours, at most one of them bypasses e; the other one can be shortcut by following P. That is, by subpath optimality, the subpath of P connecting the detour endpoints is no longer than the detour. ∎

Definition 4.3 *A* single-detour path-comparison based algorithm *for the replacement paths problem considers only subpaths of paths with at most one detour.*

All the algorithms for the replacement paths problem that we know are single-detour algorithms. This is because, as Lemma 4.2 shows, a multiple-detour path is guaranteed not to be (a subpath of) a shortest replacement path for any edge on P. More generally, all the shortest path algorithms that we know exploit local optimality: suppose π_1 and π_2 are two paths between u and v, the algorithm has determined that $\|\pi_1\| < \|\pi_2\|$, and both paths satisfy any other criteria imposed by the problem. Then any path the algorithm later constructs that includes a u–v subpath will not use π_2. Our focus on single-detour paths essentially disallows such fruitless comparisons.

Lemma 4.4 *Consider a single-detour path-comparison based algorithm that solves the replacement paths problem for the directed graph G, and suppose the algorithm does not consider a triple $(x_{a^*}, y_{b^*}, z_{c^*})$ in the subgraph H. If we modify the weights of some of the edges of the subgraph H as in Eqs. 3 and 4, then this triple becomes a subpath of some replacement path, and all other subpaths of G considered by the algorithm maintain their relative order.*

Proof: Each path can be divided into three parts: edges of P, edges of H, and edges connecting P and H. The weight from edges of P is zero; the weight from edges of H is at most $w = W/10$; and the total weight of the connector edges is some multiple of W. (The bound on the weight of the H edges follows because the algorithm examines only single-detour paths.) If the two paths have different weights from their connector edges, then their relative order is determined by these edges only—the modification to H changes any path length by less than $w = W/10$. If the two paths have equal connector edge weights, then their relative order is determined by the edges of H they contain, and the Karger-Koller-Phillips argument shows that their relative order is unchanged (since H is visited at most once by a single-detour path). ∎

Combining the results of Theorem 3.1 and Lemma 4.4, we get our main result.

Theorem 4.5 *For any n and m, with $m = O(n\sqrt{n})$, there exists a directed graph G with n nodes, m non-negatively weighted edges, and two specified vertices s and t such that any single-detour path-comparison based algorithm that solves the replacement paths problem for s, t in G must spend $\Omega(m\sqrt{n})$ time.*

Our lower bound applies to the node version of the replacement paths problem as well: we can convert our lower bound construction into a node-based problem by adding an extra node in the middle of each edge of P. Finding replacement paths for these nodes would solve our edge replacement problem, and hence both problems are subject to the same lower bound.

5 Lower Bounds on Related Problems

5.1 k shortest paths

We mentioned in the introduction the problem of finding the k shortest simple paths between a pair of nodes in a directed graph. The best algorithm for this problem dates back to Yen [24] in 1971. The worst-case complexity of his algorithm, using modern data structures, is $O(kn(m + n \log n))$. In essence, his algorithm performs $\Theta(n)$ single-source shortest path computations for each of the k shortest paths. By contrast, the undirected version of the same problem can be solved in time $O(k(m + n \log n))$ [12, 14].

Yen's algorithm, as well as those of Lawler [16], Katoh, Ibaraki, and Mine [14], and the various heuristic improvements to Yen's algorithm [6, 10, 12, 18, 19, 23] work by solving multiple instances of the replacement paths problem. The basic idea behind these algorithms is the observation that the ith shortest path must differ from each of the first $(i - 1)$ shortest paths in at least one edge. Therefore, they generate new candidate paths by solving the replacement paths problem for the previously generated shortest paths.

Our lower bound for the replacement paths problem shows that any algorithm for the k shortest paths problem that uses the replacement paths subroutine is subject to the $\Omega(m\sqrt{n})$ lower bound. Thus, if a faster algorithm for the k shortest paths problem is to emerge, it must compute the *best replacement path* without computing *all* replacement paths. That is, it must find a way to compute $\min_i d(s, t; G \setminus e_i)$ *without* having to compute all $d(x, y; G \setminus e_i)$.

5.2 Length times hop count metric

In some applications, shortest paths with fewer hops are preferable. There are multiple ways to combine the two measures: a classical way to break ties between equal length shortest paths is to choose one with fewer hops. However, a more complex metric is to rank paths by the *product* of their *length* and *number of hops*. Archer and Tardos [2, 3] suggested the use of this metric in their study of frugal path mechanism design. Their motivation stems from a negative result proved in their paper: any reasonable mechanism that motivates the owners of links in a network to bid truthfully is bound to pay a huge premium in the worst case. The premium is proportional to the number of hops in the shortest path, and so paths that minimize $\langle \text{length} \rangle \times \langle \text{hop count} \rangle$ are likely to perform better. When using the Vickrey-Clark-Grove 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.

One difficulty with the $\langle \text{length} \rangle \times \langle \text{hop count} \rangle$ metric is that it *does not respect subpath optimality.* We need to extend our lower bound construction slightly to extend the argument to this metric. We prepend a path of $2n$ zero-weight edges to the shortest path P in our lower bound construction of Section 2, and move s to the beginning of this prefix path. Let us call this graph G_1; it still has $O(n)$ vertices and $O(m)$ edges. We note that the replacement cost for e_i, where $i = 1, 2, \ldots, 2n$, is *infinite*; we are only interested in the replacement costs for the edges e_i, for $i = 2n+1, \ldots, 3n$.

Lemma 5.1 *Although subpath optimality does not hold for the $\langle$length$\rangle \times \langle$hop count$\rangle$ metric, nevertheless in the graph G_1 constructed above, the replacement path for each e_i, where $i = 2n+1, \ldots, 3n$, makes exactly one detour. Furthermore, the shortest path from s to t in the graph $G_1 \setminus (v_i, v_{i+1})$, where $i = 2n+1, \ldots, 3n$, has cost*

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

where the edge (v_i, v_{i+1}) corresponds to the pair (s_j, t_j) in H.

Proof: The key observation is that if a replacement path makes d detours, with detour r skipping k_r edges of the path P, then the weight of this replacement path equals

$$(3n - \sum_r k_r + 4d) \times \left((dn + \sum_r k_r)W + \sum_r \ell_r \right),$$

where the first term is the number of hops in the path, and the second term is the length of the path; the term $\ell_r \leq w$ is the length of the subpath in detour r that lies in H. Since this function grows monotonically with d, the replacement cost is minimized for $d = 1$. With a single detour that bypasses k edges of the path P, a replacement path π has exactly $(3n - k + 4)$ hops and its length satisfies $(n+k)W < \|\pi\| < (n+k+0.1)W$. It is easy to see that the product of length and hops is minimized by choosing k as small as possible, namely $k = 1$. ∎

We conclude that the replacement paths problem with the $\langle \text{length} \rangle \times \langle \text{hop count} \rangle$ metric also has a lower bound of $\Omega(m\sqrt{n})$ time in the single-detour path comparison model.

5.3 Edge Replacement Shortest Path Trees

Suppose $T(s)$ is the shortest path tree in G rooted at s. We wish to determine how the tree changes if we remove one edge of $T(s)$. In particular, let $T_{-e}(s)$ denote the shortest path tree rooted at s in the graph $G \setminus e$. We wish to compute $T_{-e}(s)$ for all edges $e \in T(s)$.

We create a path $s = v_1, \ldots, v_n = t$, where each edge has cost 0, and link it to an instance of the Karger-Koller-Phillips lower bound construction for the all pairs shortest paths problem. Let $U = \{u_1, u_2, \ldots, u_n\}$ be the set of nodes in the class U. We create a directed edge from v_i to u_i, with cost $(n-i)W$. We claim that all the n^2 shortest path distances in the lower bound subgraph can be deduced by computing the edge replacement shortest path tree $T_{-e}(s)$, where $e = (v_i, v_{i+1})$, for $i = 1, 2, \ldots, n-1$. We omit the details from this abstract, and simply state the main result.

Theorem 5.2 *Given a directed graph G with non-negative edge weights, the problem of computing edge replacement shortest path trees for a fixed source s requires $\Omega(nm)$ time in the path comparison model. This bound is nearly tight, since the problem can be solved in $O(n(m + n\log n))$ time in the same model.*

6 Upper Bounds

The best upper bound known for the replacement paths problem is $O(n(m + n \log n))$. The best upper bound for the k shortest simple paths problem is $O(kn(m + n \log n))$. Thus, there remains a substantial gap between our lower bound and the upper bounds. However, for the specific problem instances constructed for our lower bounds, we can solve the n-pairs shortest paths subproblem in $O(\sqrt{n}(m + n \log n))$ time—we run a single-source shortest path algorithm from each x_a node. Once all the $O(n)$ x–z distances in the subgraph H have been computed, we show that the replacement paths (for the shortest path metric) can be computed in additional $O(n\sqrt{n} \log n)$ time.

We present our upper bound for a slight generalization of the lower bound construction. Suppose that every detour from the path P passes through at least one of a set of $O(\sqrt{n})$ pre-specified *gateway* vertices not on P. We compute the distances from each gateway to all the vertices of P in the graph $G \setminus P$, that is, using only paths disjoint from P until they reach P. Likewise, we compute the distances from all the vertices of P to the gateways, using paths in $G \setminus P$. (This is done by finding shortest paths *from* the gateways in a graph obtained from $G \setminus P$ by reversing all the edges.) These path computations take $O(\sqrt{n}(m + n \log n))$ time. In additional $O(n\sqrt{n})$ time, we can compute, for each gateway g and each edge $e \in P$, the length of the shortest path from s to g that leaves P before e, and the length of the shortest path from g to t that first touches P after e. The sum of these distances is the length of the shortest replacement path for e that passes through g. Minimizing over all gateways g gives the overall shortest replacement path for e. (If the graph also happens to have some single-edge detours, which necessarily do not pass through any gateway, we can find the shortest replacement paths that use these single-edge detours in $O(m + n \log n)$ time [11].)

7 Concluding Remarks

We have shown that the replacement paths problem in a directed graph with $m = O(n\sqrt{n})$ edges has a lower bound of $\Omega(m\sqrt{n})$ for any single-detour path-comparison based algorithm. Any algorithm for the k shortest simple paths in a directed graph that computes replacement paths is also subject to this lower bound. In fact, the lower bound holds even for computing the second shortest path. To the best of our knowledge, these are the first super-linear lower bounds for these problems. By contrast, both of these problems can be solved in (near) linear time for undirected graphs. We also established a nearly tight lower bound of $\Omega(nm)$ for the edge-deletion problem in *shortest path trees*.

A worthwhile distinction to make is between directed acyclic graphs (DAGs) and general directed graphs. The lower bound of Karger, Koller, and Phillips [13] holds even for DAGs. On the other hand, both the replacement paths problem and the k shortest paths problems can be solved much more efficiently for DAGs. In particular, our undirected graph algorithms work for DAGs, so the replacement paths problem can be solved in $O(m + n \log n)$ time for DAGs [11]; the k shortest paths problem can be solved in $O(k + m + n \log n)$ time by a method not based on replacement paths [7].

References

[1] N. Alon, Z. Galil and O. Margalit. On the Exponent of the All Pairs Shortest Path Problem. In *32nd Symposium on Foundations of Computer Science*, 569–575, 1991.

[2] A. Archer. Private communication, 2001.

[3] A. Archer and E. Tardos. Frugal path mechanisms. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 991–999, 2002.

[4] M. O. Ball, B. L. Golden, and R. V. Vohra. Finding the most vital arcs in a network. *Oper. Res. Letters*, 8:73–76, 1989.

[5] A. Bar-Noy, S. Khuller, and B. Schieber. The complexity of finding most vital arcs and nodes. Technical Report CS-TR-3539, Institute for Advanced Studies, University of Maryland, College Park, MD, 1995.

[6] A. Brander and M. Sinclair. A comparative study of K-shortest path algorithms. In *Proc. of 11th UK Performance Engineering Workshop*, pages 370–379, 1995.

[7] D. Eppstein. Finding the k shortest paths. *SIAM J. Computing*, 28(2):652–673, 1998.

[8] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *INFOCOM*, pages 519–528, 2000.

[9] M. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. of Computing*, 5:83–89, 1976.

[10] E. Hadjiconstantinou and N. Christofides. An efficient implementation of an algorithm for finding K shortest simple paths. *Networks*, 34(2):88–101, September 1999.

[11] J. Hershberger and S. Suri. Vickrey prices and shortest paths: What is an edge worth? In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, pages 252–259, 2001.

[12] J. Hershberger, S. Suri, and A. Bhosle. Finding the k shortest simple paths. Technical report, University of California, Santa Barbara, 2001.

[13] D. R. Karger, D. Koller, and S. J. Phillips. Finding the hidden path: Time bounds for all-pairs shortest paths. *SIAM J. Comput.*, 22:1199–1217, 1993.

[14] N. Katoh, T. Ibaraki, and H. Mine. An efficient algorithm for k shortest simple paths. *Networks*, 12:411–427, 1982.

[15] D. Kirkpatrick and S. Reisch. Upper bounds for sorting integers on random access machines. *Theoretical Computer Science*, 28(3):263–276, 1984.

[16] E. L. Lawler. A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, pages 401–405, 1972.

[17] K. Malik, A. K. Mittal, and S. K. Gupta. The k most vital arcs in the shortest path problem. *Oper. Res. Letters*, 8:223–227, 1989.

[18] E. Martins and M. Pascoal. A new implementation of Yen's ranking loopless paths algorithm. Submited for publication, Universidade de Coimbra, Portugal, 2000.

[19] E. Martins, M. Pascoal, and J. Santos. A new algorithm for ranking loopless paths. Technical report, Universidade de Coimbra, Portugal, 1997.

[20] E. Nardelli, G. Proietti, and P. Widmayer. Finding the most vital node of a shortest path. In *Proc. COCOON*, 2001.

[21] N. Nisan and A. Ronen. Algorithmic mechanism design. In *Proc. 31st Annu. ACM Sympos. Theory Comput.*, 1999.

[22] W. J. Paul and J. Simon. Decision trees and random access machines. In *Proc. International Symp. on Logic and Algorithmic*, pages 331–340, 1980.

[23] A. Perko. Implementation of algorithms for K shortest loopless paths. *Networks*, 16:149–160, 1986.

[24] J. Y. Yen. Finding the K shortest loopless paths in a network. *Management Science*, 17:712–716, 1971.

[25] 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.

[26] U. Zwick. All Pairs Shortest Paths in Weighted Directed Graphs—exact and almost exact algorithms. In *Proc. IEEE Symposium on Foundations of Computer Science*, 310–319, 1998.

[27] U. Zwick. All Pairs Shortest Paths using Bridging Sets and Rectangular Matrix Multiplication. In *Electronic Colloquium on Computational Complexity*, 2000.