Replacement Paths for Pairs of Shortest Path Edges in Directed Graphs

AMIT M. BHOSLE*
YASU Technologies Pvt. Ltd.
1-10-98/16, Raj Plaza, Dwarkadas Colony
Begumpet, Hyderabad - 500 016, India
email: bhosle@cs.ucsb.edu

TEOFILO F. GONZALEZ
Department of Computer Science
University of California
Santa Barbara, CA 93106, USA
email: teo@cs.ucsb.edu

ABSTRACT

The (Single-Edge) Replacement Paths *problem is defined as follows: Given a weighted graph $G(V, E)$, two nodes s and t, and the shortest path $\mathcal{P}_G(s,t) = \{e_1, e_2, \ldots, e_p\}$ from s to t in G, compute the shortest path from s to t in the graph $G \backslash e_i$ for $1 \leq i \leq p$. In other words, the single-edge replacement paths problem studies how a given s-t shortest path changes with the deletion of an edge lying on the path. We study a* two-edge *generalization of this problem for* directed *graphs, termed the* Edge Pairs Replacement Paths *problem: Given G, s, t, and $\mathcal{P}_G(s,t)$ as defined above, compute the shortest path from s to t when two edges of $\mathcal{P}_G(s,t)$ fail, for all the p^2 pairs of edges of $\mathcal{P}_G(s,t)$. We present an $O(n^3)$ algorithm for this problem, and establish an $\Omega(mn)$ lower bound in the* path comparison *model for shortest path algorithms which was introduced in* [*10*].

Our algorithm is based on a new algorithm for the directed version of the single-edge replacement paths problem which makes clever use of the information provided by the all-pairs-shortest-paths computation on a modification of the input graph and avoids extensive recomputations required by the naive algorithm.

1 Introduction

Shortest paths computation is one of the most fundamental problems in graph theory. The huge interest in the problem is mainly due to the wide spectrum of its applications, ranging from routing in communication networks to robot motion planning, scheduling, sequence alignment in molecular biology and length-limited Huffman coding, to name only a very few.

In several scenarios, one needs to compute not only the shortest path between two given nodes, but also several others which satisfy certain additional criteria. Routing in networks which are susceptible to link failures is one such example. When some links of the network fail, routes need to be found which do not use the failed link(s). Though, *single* link failures has been widely studied, in certain situations, it becomes important to consider the failure of at least two links at the same time. As mentioned in [2], in some cases, it takes a significant amount of time (anywhere from a few hours to a few days) to repair a failed link, during which another link of the network may fail. Another scenario arises when two (or more) links of a network share some portion in the physical design and a fault in the shared region renders all the links involved as failed.

We study the following shortest paths problem, termed the *Edge Pairs Replacement Paths* problem on directed graphs: Given a directed weighted graph $G(V, E)$, two nodes s and t, and the shortest path $\mathcal{P}_G(s,t) = \{e_1, e_2, \ldots, e_p\}$ from s to t in G, compute the shortest path from s to t when 2 edges of $\mathcal{P}_G(s,t)$ fail, for all the p^2 pairs of edges of $\mathcal{P}_G(s,t)$. In essence, we study how a given s-t shortest path changes with the failure of any two edges of the path. This is a *two-edge* generalization of the better studied (*single-edge*) *replacement paths* problem discussed later in Section 1.1.

Our solution to the edge pairs replacement path problem lends itself to a solution for determining the Two-Most-Vital-Arcs of a shortest path. A pair of edges (arcs) lying on an s-t shortest path is termed the two most vital arcs of the path if the deletion of these two edges produces the largest increase in the s-t shortest path distance as compared to any other pair of edges lying on the path.

A naive algorithm for the Edge Pairs Replacement Paths problem is based on recomputation: For every pair of edges $\{e, f\} \subseteq \mathcal{P}_G(s,t)$, compute the shortest path from s to t in the graph $G \backslash \{e, f\}$, where $G \backslash \{e, f\}$ represents the graph G with the edges e and f removed. Since $\mathcal{P}_G(s,t)$ can have as many as $O(n)$ edges, there can be as many as $O(n^2)$ pairs of edges lying on $\mathcal{P}_G(s,t)$. Thus, the naive algorithm takes $O(n^2(m + n \log n))$ time using the F-Heap[5] implementation of Dijkstra's single source shortest path algorithm[3].

1.1 Related Work

The (*single-edge*) *replacement paths* problem basically studies how a given s-t shortest path changes if an edge of the path fails. Formally, the problem is defined as follows: Given a graph $G(V, E)$, two nodes $s, t \in V$, and a shortest path $\mathcal{P}_G(s,t) = \{e_1, e_2, \ldots, e_p\}$ from s to t in G, compute the shortest path from s to t in each of the p graphs $G \backslash e_i$

*This work was done while this author was at the Department of Computer Science, University of California, Santa Barbara, CA - 93106.

for $1 \leq i \leq p$, where $G \backslash e_i$ represents the graph G with the edge e_i removed.

For *undirected* graphs, (near) optimal algorithms for this problem have been around for a while. Malik, Mittal and Gupta[11] presented an $O(m+n \log n)$ time algorithm for finding the most-vital-arc with respect to an s-t shortest path [1].

The replacement paths problem was also proposed later by Nisan and Ronen[12] in their work on Algorithmic Mechanism Design. Hershberger and Suri[8] rediscovered the algorithm of [11] in their work in the domain of algorithmic mechanism design related to computing the *Vickrey payments* for the edges lying on an s-t shortest path. Note that the undirected version can trivially be solved in $O(mn + n^2 \log n)$ time using known results for the single-edge version of the problem.

The replacement paths problem on *directed* graphs was shown to admit a lower bound of $\Omega(min(n^2, m\sqrt{n}))$ for a certain class of path-comparison based algorithms for shortest paths by Hershberger, et. al.[9]. The path comparison model for shortest path algorithms was introduced in [10] and further explored in [9, 6]. It forms a natural class for such algorithms. In this model, an algorithm determines the shortest paths by comparing the lengths of two different paths. Such an algorithm can perform all standard operations in unit time, but its access to edge weights is only through the comparison of two paths. Most of the known shortest paths algorithms fall into this category, including those by Dijkstra, Floyd, Spira[13], Frieze-Grimmet and the hidden paths algorithm of [10]. However, algorithms using fast matrix multiplication like those of Fredman[4] and Takaoka[14] do not fall into this category since they also add up weights of edges that do not form a path. The reader is referred to [10, 9] for further details of the model of computation.

1.2 Main Results

We present an $O(n^3)$ algorithm for the Edge Pairs Replacement Paths problem on directed graphs. We also establish a lower bound of $\Omega(mn)$ in the path-comparison model of [10] for shortest paths algorithms.

Our algorithm for the $EPRP$ problem is based on a new algorithm for the replacement-paths-problem on directed graphs. This algorithm makes clever use of the information provided by an *all-pair-shortest-paths* computation on a *modification* of the input graph G and avoids the $O(n)$ single-source shortest paths computations employed by the naive algorithm. Consequently, our replacement paths algorithm performs favorably as compared to the naive algorithm whenever the all-pair-shortest-paths (APSP) computation is faster than n single-source shortest-paths computations. Thus, while the naive algorithm requires $\Theta(n^3)$ time in the worst case, our algorithm is the first subcubic algorithm for this problem since the time required for an all-pair-shortest-paths computation is subcubic in quite a few situations, as will be discussed in Section 3.

[1]The algorithm presented in [11] had a minor flaw which was subsequently pointed out and corrected in [1]

2 Preliminaries

Our communication network is modeled by a weighted directed graph $G(V, E)$, with $n = |V|$ and $m = |E|$. Each edge $e \in E$ has an associated cost, $cost(e)$, which is a real number. It is assumed that the shortest paths in G are *well defined*. That is, there are no *negative cycles* in G.

We use $\mathcal{P}_G(x, y)$ to denote the shortest path from a node $x \in G(V)$ to a node $y \in G(V)$, and $d_G(x, y)$ to denote its weight. We drop the subscript G when it is clear the graph we refer to. Note that $\mathcal{P}_G(s, t)$ can equivalently be represented as a sequence of edges or as a sequence of vertices lying on it. $|\mathcal{P}|$ is used to denote the total weight of a path $\mathcal{P}$ in the graph.

$T_{apsp}(m, n)$ represents the time required for an all-pair-shortest-paths computation on a directed graph with m edges and n vertices.

A cut in a graph is the partitioning of the set of vertices V into V_1 and V_2 and it is denoted by (V_1, V_2). The set of edges $E(V_1, V_2)$ represents all the edges across the cut (V_1, V_2). We use $\mathcal{T}_s$ to denote the tree of shortest paths from a node $s \in V$ to all other nodes in V. $G \backslash X$ denotes the graph G with the elements in the set X removed from G.

$\mathcal{R}_G(s, t; e)$ is used to denote the replacement path for the edge e, which is defined as the shortest path from s to t in the graph $G \backslash e$. When G, s and t are clear from the context, we use only $\mathcal{R}(e)$ to denote $\mathcal{R}_G(s, t; e)$.

Finally, for simplicity we assume that the graph is robust enough to tolerate the link failures. In our case, this translates to the assumption that for any two edges $e, f \in \mathcal{P}_G(s, t)$, there exists a path from s to t in $G \backslash \{e, f\}$. Note that this assumption only makes the explanation simpler and is not actually critical for the algorithms.

2.1 Organization of the Paper

We present the $O(T_{apsp}(m, n))$ time algorithm for the replacement paths problem on directed graphs in Section 3. In Section 4.2, we present an $O(n^3)$ algorithm for the Edge Pairs Replacement Paths problem, which borrows critical observations and ideas from the replacement paths algorithm. The $\Omega(mn)$ lower bound (in the path-comparison model for shortest paths algorithms) for the Edge-Pairs-Replacement-Paths problem is presented in Section 4.1. Finally, we conclude the paper with some open problems in Section 5.

3 Directed Replacement Paths

In this section we present an algorithm for the replacement paths problem on directed graphs which requires

$O(T_{apsp}(m,n))$ time. That is, the time required by this algorithm is dominated by one *all-pairs-shortest-paths* computation. This algorithm makes clever use of the information provided by an all-pairs-shortest-paths computation on a *modification* of the input graph G and avoids the $O(n)$ *single-source-shortest-paths* computations employed by the naive algorithm. Thus, this algorithm performs favorably as compared to the naive algorithm whenever the all-pair-shortest-paths computation is faster than $O(n)$ single-source-shortest-paths computations, which happens in quite a few cases: Takaoka[14] showed that $T_{apsp}(m,n) = O(n^3\sqrt{\log\log n/\log n})$. If edge weights are integers, Zwick's algorithm[15] for the APSP takes $O(C^{0.681}n^{2.575})$ time, where C is the heaviest edge of the graph. The hidden paths algorithm of [10] takes $O(m^*n + n^2\log n)$ time where m^* is the number of edges participating in the shortest paths. As shown in [10], m^* is likely to be small in practice, since $m^* = O(n\log n)$ with high probability for many probability distributions on edge weights. Spira's[13] all pair shortest paths algorithm takes $O(n^2\log^2 n)$ average time. An appropriate APSP algorithm can be chosen to improve upon the cubic naive bound.

3.1 Description of the Algorithm

We start off by computing the shortest paths tree, $\mathcal{T}_s$, of s and the All Pairs Shortest Paths in the graph $G' = G\backslash\mathcal{P}_G(s,t)$ obtained from G by deleting all the edges lying on $\mathcal{P}_G(s,t)$, the shortest path from s to t.

Note that deleting any edge e_i on $\mathcal{P}_G(s,t)$ splits $\mathcal{T}_s$ in two components which induces a cut in the graph G. In this cut, denote by $V_{s|i}$ the component containing s. Consider the case when an edge $e_i \in \mathcal{P}_G(s,t)$ is deleted (See Figure 1). The weight of the shortest path through a node $v \in V_{s|i}$ which touches the spine $\mathcal{P}_G(s,t)$ at a node u, and does not pass through any other vertex of $V_{s|i}$ is

$$d_{G'}(s \rightsquigarrow v \rightsquigarrow u \rightsquigarrow t) = d_G(s,v) + d_{G'}(v,u) + d_G(u,t) \quad (1)$$

where $d_{G'}(x,y)$ and $d_G(x,y)$ represent the weights of the paths $\mathcal{P}_{G'}(x,y)$ and $\mathcal{P}_G(x,y)$ respectively. The first and third terms in the above expression are obtained from $\mathcal{T}_s$ while the second one was computed in the APSP computation.

The following lemma captures critical properties of $\mathcal{R}_{e_i}$, the replacement path of the edge $e_i \in \mathcal{P}_G(s,t)$. For brevity, we omit the proof.

Lemma 3.1 *The replacement path for an edge $e_i \in \mathcal{P}_G(s,t)$, $\mathcal{R}_{e_i}$, which is defined as the shortest path from s to t in $G\backslash e_i$, follows the path in $\mathcal{T}_s$ from s to some node $v \in V_{s|i}$, uses exactly one edge across the cut (induced in G by the two components of $\{\mathcal{T}_s\backslash e_i\}$) to join the spine $\mathcal{P}_G(s,t)$ at some vertex u, and follows the spine all the way upto t.*

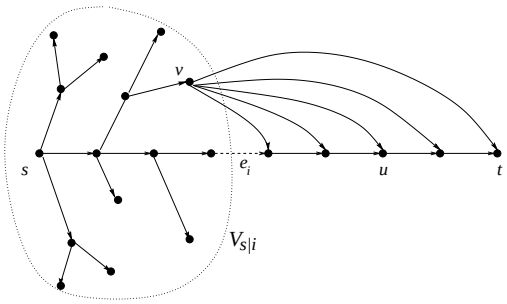

Figure 1. Candidates for the best replacement path through a node $v \in V_{s|i}$

If $\mathcal{P}_G(s,t) = \{s = u_0, u_1, \ldots, u_k = t\}$, the best replacement path through a node $v \in V_{s|i}$ would be:

$$|\mathcal{R}^v_{e_i}(s,t)| = MIN^k_{j=i+1}\{d_G(s,v) + d_{G'}(v,u_j) + d_G(u_j,t)\}$$

where $e_i = (u_i, u_{i+1})$ Note that the paths involved in the above minimization *do not* use the edge e_i, owing to the construction of G'. Thus the path $\mathcal{R}^v_{e_i}(s,t)$ defined above is a candidate for the replacement path for e_i. Finally, the best replacement path for the edge e_i would be the shortest one among all such candidate replacement paths through the nodes in $V_{s|i}$. That is,

$$|\mathcal{R}(e_i)| = |\mathcal{P}_{G\backslash e_i}(s,t)| = MIN_{v\in V_{s|i}}\{|\mathcal{R}^v_{e_i}(s,t)|\} \quad (2)$$

Note that the subpath $(v \rightsquigarrow t)$ of $\mathcal{R}^v_{e_i}(s,t)$ may not be the shortest path from v to t since for computing $\mathcal{R}^v_{e_i}(s,t)$ we only consider paths which do not use any other vertex of $V_{s|i}$. However, this does not pose a problem since we iterate over all vertices of $V_{s|i}$ for determining $|\mathcal{R}_{e_i}|$ and the node $v \in V_{s|i}$ which minimizes the expression is indeed the last vertex of $V_{s|i}$ that lies on the path (Lemma 3.1). Note that the equation (1) is pretty straight-forward and even resembles the one used in [11, 8]. However, as shown in [7], that equation alone does not suffice to solve the problem for directed graphs, for which extensions to equation (2) were necessary.

An arbitrary way of computing these values may not result in much of an improvement. We start in an orderly fashion: from the head, t, of $\mathcal{P}_G(s,t)$, the reason being that this allows updating the $|\mathcal{R}^v_{e_i}(s,t)|$ values efficiently. When computing the replacement path for an edge $e_i = (u_i, u_{i+1})$, we update this value for all $v \in V_{s|i}$ as

$$|\mathcal{R}^v_{e_i}(s,t)| = MIN\{|\mathcal{R}^v_{e_{i+1}}(s,t)|, d_G(s,v) + d_{G'}(v,u_{i+1}) + d_G(u_{i+1},t)\} \quad (3)$$

where the second term is the weight of the path $(s \rightsquigarrow v \rightsquigarrow u_{i+1} \rightsquigarrow t)$.

We formally present the algorithm *Directed* (*Single-Edge*) *Replacement Paths* below.

`Algorithm DSERP`

- Input: Directed weighted graph $G(V, E)$, two specified nodes s and t, and a shortest path from s to t in G, $\mathcal{P}_G(s,t) = \{e_1, e_2, \ldots, e_p\}$.
- Initialization: Compute the shortest paths tree, $\mathcal{T}_s$ of s and compute the all-pairs-shortest-paths in the graph G' which is obtained by deleting from G, all the edges of $\mathcal{P}_G(s,t)$. Compute $V_{s|p}$ as described above. Also, for each node $v \in V_{s|p}$, compute $|\mathcal{R}^v_{e_p}(s,t)|$. Report $|\mathcal{R}(e_p)| = MIN_v\{\mathcal{R}^v_{e_p}(s,t)\}$.
- For $i = p-1$ to 1, do:
  - Compute $V_{s|i}$ by a traversal of $\mathcal{T}_s \backslash e_i$.
  - For each $v \in V_{s|i}$, update $|\mathcal{R}^v_{e_i}(s,t)|$ according to equation (3).
  - Report $|\mathcal{R}(e_i)| = MIN_v\{\mathcal{R}^v_{e_i}(s,t)\}$.

The time complexity is straight forward: $|V_{s|i}|$ is at most $O(n)$ for any deleted edge $e_i \in \mathcal{P}_G(s,t)$. Also, the set $V_{s|i}$ can be computed in $O(n)$ time using any standard traversal (breadth-first-traversal, depth-first-traversal, etc) on $\mathcal{T}_s \backslash e_i$. Updating the $|\mathcal{R}^v_{e_i}(s,t)|$ values requires constant time per node $v \in V_{s|i}$, per deleted edge $e_i \in \mathcal{P}_G(s,t)$. Since $\mathcal{P}_G(s,t)$ 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 remains dominated by the APSP computation.

We have thus established the following theorem:

Theorem 3.1 *Every instance of the directed version of the (single-edge) replacement paths problem can be solved in time $O(T_{apsp}(m,n))$, where $T_{apsp}(m,n)$ denotes the time required to perform an all-pairs-shortest-paths computation on a graph.*

Proof: The correctness of the algorithm DSERP and the bound on its running time follow from the discussion immediately preceding the algorithm. □

4 Edge Pairs Replacement Paths in Directed Graphs

In this section, we discuss the Edge Pairs Replacement Paths problem on directed graphs. As mentioned earlier, a naive algorithm for the problem takes $O(n^2(m + n\log n))$ time. In sub-section 4.1, we establish an $\Omega(mn)$ lower bound for the problem in the *path-comparison* model for shortest paths algorithms introduced by Karger, et. al.[10] and further explored in [6, 9]. The $O(n^3)$ algorithm for the problem is presented in sub-section 4.2.

4.1 Lower Bound

The lower bound is basically a reduction of an instance of the all pairs shortest paths (APSP) problem to an instance of the Edge-Pairs-Replacement-Paths problem. The reduction can be performed in linear time by adding some nodes and edges to the input graph H of the APSP instance. The modified graph obtained from H is called G. As we shall see later, G also contains $O(n)$ vertices and $O(m)$ edges, where n and m represent the number of vertices and edges in H.

We use the lower bound of $\Omega(mn)$ on the APSP problem presented in [10]. This bound holds in the path-comparison model for shortest path algorithms. The reader is referred to [10, 9] for details of the model of computation. We briefly outline the APSP lower bound of [10] here. The graph H used in the construction of [10] has a total of mn directed paths from all the source vertices to all the destination vertices. The basis of the lower bound proof is that any algorithm must include all of these mn paths in the computation. If an algorithm fails to include any of these paths, say (S_i, V_k, D_j) in its computation, then the weights of some of the edges of H (the thicker ones in the figure) can be modified such that:

(1) The ignored path, (S_i, V_k, D_j), becomes the shortest path from S_i to D_j.

(2) All other paths retain the same relative order of weights as initially.

Thus, after the weight modification, the algorithm cannot

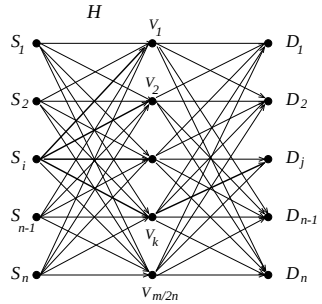

Figure 2. Construction for APSP Lower Bound used in [10]

output the correct answer for the (S_i, D_j) distance since it ignores the particular path and all other path comparisons will yield the same information as earlier.

We use the same graph as used in [10] as our graph H (see Figure 2). H is a tri-partite graph and has n vertices in the first and third column, and $m/2n$ vertices in the middle column. There are $m/2$ edges from vertices of the first column to those of the second, and $m/2$ from the vertices of the second column to those of the third. Thus, H has $3n$ vertices and m edges. As shown in [10], any path comparison based shortest path algorithm must spend at least $\Omega(mn)$ time to compute the shortest paths from all the vertices in the first column to all the vertices in the third column. Let us denote the n source vertices in H by $S_1, S_2, \ldots, S_n$ and the destination vertices by

$D_1, D_2, \ldots, D_n$.

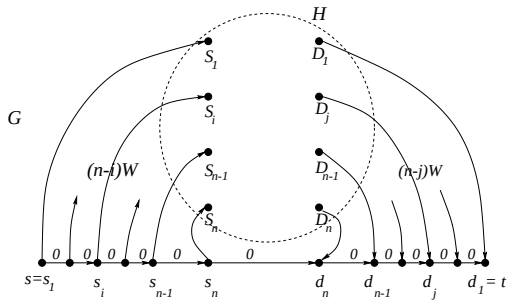

Figure 3. Construction for Edge-Pairs-Replacement-Paths Lower Bound

To reduce this problem to an instance of the Edge-Pairs-Replacement-Paths problem, we create a spine of $2n$ vertices: $P = \{s_1, s_2, \ldots, s_n, d_n, d_{n-1}, \ldots, d_1\}$. Each edge of the spine has weight 0. We map all the s_i's to the corresponding S_i's in H and all the d_i's to the corresponding D_i's in H. Also, we choose a quantity $W = 10{\cdot}n{\cdot}e_{max}$ where e_{max} is the weight of the heaviest edge of H. Note that this choice of W makes it much more heavier than the weight of any simple path in H since any simple path in H can have weight at most $(n-1) \cdot e_{max}$. Finally, we introduce the following edges to connect the spine P to H:
(1) For $1 \leq i \leq n$, an edge from s_i to S_i of weight $(n-i)W$.
(2) For $1 \leq i \leq n$, an edge from D_i to d_i of weight $(n-i)W$.
This entire graph is called G. For the s and t vertices in the Edge-Pairs-Replacement-Paths problem instance, we choose $s = s_1$ and $t = d_1$. The set F contains the edges on the spine. The paths from s to t in in this construction satisfy the following property. Note that the formulae for the edge weights are minor variants of those used in the lower bound construction of [9].

Lemma 4.1 *The shortest path from s to t when the edges (s_i, s_{i+1}) and (d_{j+1}, d_j) are removed from the graph G, has weight exactly equal to*

$$d_{G\setminus\{(s_i,s_{i+1}),(d_{j+1},d_j)\}}(s,t) = (2n-i-j){\cdot}W + d_H(S_i, D_j)$$

where $d_H(S_i, D_j)$ represents the weight of the shortest path from S_i to D_j in the graph H.

Proof: First note that the weight of the path that starts from s, follows P to s_i, enters H at S_i using the edge (s_i, S_i), takes the shortest path from S_i to D_j, returns to P using the edge (D_j, d_j), and follows the spine to t has weight precisely given by the expression above. Let us call this path π. We now show that any other path from s to t in $G\setminus\{(s_i, s_{i+1}), (d_{j+1}, d_j)\}$ is heavier than π. Consider the paths that leave the spine at a vertex s_p and returns at d_q with either $p < i$ or $q < j$. If $p < i$, this path has weight at least $(n-p){\cdot}W + (n-j){\cdot}W \geq (2n-i-j){\cdot}W + W > |\pi|$ since $W > d_H(x, y)$ for any $x, y \in H$. Similarly, any path with $q < j$ is heavier than π. This completes the proof. □

The above lemma shows that a solution for the Edge-Pairs-Replacement-Paths problem instance constructed above provides enough information to quickly compute the all pair shortest paths distances for the graph H. In other words, an $O(f(m,n))$ time solution for the Edge-Pairs-Replacement-Paths problem instance G implies an $O(f(m,n))$ time solution for the APSP problem instance H. But, as seen from the APSP lower bound of [10], any algorithm for the latter must spend at least $\Omega(mn)$ time. Thus, we conclude that any algorithm for the Edge-Pairs-Replacement-Paths problem must spend at least $\Omega(mn)$ time. The formal argument is similar to the one in [10]: to compute the shortest path from s to t when the edges (s_i, s_{i+1}) and (d_{j+1}, d_j) are removed, the algorithm must compare all the $m/2n$ paths between S_i and D_j in H. Otherwise, we can modify the weights of some of the edges as in [10] and force the algorithm to report an incorrect answer. Thus, computing the shortest paths for all the $O(n^2)$ edge-pairs deletions requires the algorithm to investigate a total of $\Theta(mn)$ paths.

4.2 Upper Bound

We now describe the algorithm for the Edge-Pairs-Replacement-Paths problem. The algorithm is borrows critical ideas from the algorithm presented in Section 3 for the directed version of the (single-edge) replacement paths problem.

4.2.1 Description of the Algorithm

We describe the procedure to compute the replacement paths when all pairs of edges including a particular edge $e_i \in \mathcal{P}_G(s,t)$ fail. This procedure takes $O(n^2)$ time. The same procedure is then used for every edge $e \in \mathcal{P}_G(s,t)$, thus solving the entire instance of the Edge-Pairs-Replacement-Paths problem in $O(n^3)$ time. Apart from this, an all-pair-shortest-path computation is performed on $G\setminus\mathcal{P}_G(s,t)$ that does not affect the asymptotic worst case time bound of $O(n^3)$.

Let $E(\mathcal{P}_G(s,t)) = \{e_1, e_2, \ldots, e_p\}$ and $V(\mathcal{P}_G(s,t)) = \{s = u_0, u_1, \ldots, u_p = t\}$ be the edge and vertex sets of $\mathcal{P}_G(s,t)$.

As the first step, we perform an APSP computation on the graph $G_{-\mathcal{P}_G(s,t)} = G\setminus E(\mathcal{P}_G(s,t))$. Next, we construct the shortest paths tree of s in the graph $G_{-i} = G\setminus e_i$. Denote this tree by $\mathcal{T}_s^i$ and the path from s to t in this tree by $\mathcal{P}_{G_{-i}}(s,t)$. Let $F_i = \{f_1, f_2, \ldots, f_k\}$ denote the set of edges common in $\mathcal{P}_{G_{-i}}(s,t)$ and $\mathcal{P}_G(s,t)$. Notice that if the second edge in $\mathcal{P}_G(s,t)$ which fails is not in F_i, then the shortest path from s to t remains as defined by $\mathcal{T}_s^i$. Thus, it is only the edges of F_i which can further alter the shortest path from s to t. Note that as shown in Figure 4, F_i consists of edges from two subpaths of $\mathcal{P}_G(s,t)$: one starting from s and the other ending at t. Denote these subpaths by

F_i^s and F_i^t respectively. We ignore the edges of F_i^s: the case when the pair (e_i, f) fails such that $f \in \mathcal{P}_G(s,t) \cap F_i^s$ will be handled when the algorithm works to solve the pairs including f as one of the failed edges.

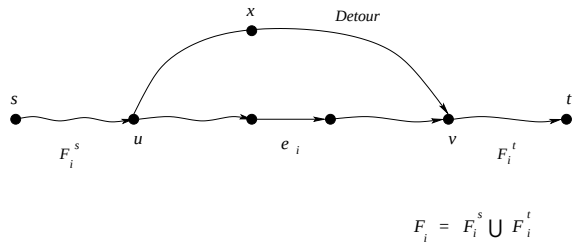

Figure 4. $\mathcal{R}_{e_i} = \mathcal{P}_{G \backslash e_i}(s,t) = (s \rightsquigarrow u \rightsquigarrow x \rightsquigarrow v \rightsquigarrow t)$ and $F_i = \mathcal{P}_G(s,t) \cap \mathcal{R}_{e_i}$

The remaining job is very similar to the replacement paths problem we discussed earlier. Consider the case when (in addition to e_i) an edge $f \in F_i^t$ fails and we need to compute the s-t shortest path in $G \backslash \{e_i, f\}$. We first construct the cut in G induced by the two components of $\mathcal{T}_s^i \backslash f$. Denote by $V_{s|\{e_i,f\}}$ the component containing s. For each node $v \in V_{s|\{e_i,f\}}$, we keep track of the shortest path from s to t through v which does not pass through any other vertex of $V_{s|\{e_i,f\}}$. In other words, v is the last vertex of $V_{s|\{e_i,f\}}$ that this path touches. Using $\mathcal{R}^v_{\{e_i,f\}}(s,t)$ to denote this path, we have

$$|\mathcal{R}^v_{\{e_i,f\}}(s,t)| = MIN^p_{k=j+1}\{ d_{G_{-i}}(s,v) + d_{G_{-\mathcal{P}_G(s,t)}}(v,u_k) + d_{G_{-i}}(u_k,t)\} \quad (4)$$

where $f = (u_j, u_{j+1})$.

Note that $d_{G_{-i}}(s,v) = d_G(s,v)$ and $d_{G_{-i}}(u_k,t) = d_G(u_k,t)$ since they are all subpaths of $\mathcal{P}_G(s,t)$ not using the edges e_i or f. Since $\mathcal{R}^v_{\{e_i,f\}}(s,t)$ as defined above does not use the edges e_i or f, it is a valid candidate for the replacement path for the edge pair $\{e_i, f\}$. Finally, the replacement path for the pair $\{e_i, f\}$ is chosen as the minimum among all candidate replacement paths. That is,

$$|\mathcal{R}(e_i,f)| = MIN_{v \in V_{s|\{e_i,f\}}}\{|\mathcal{R}^v_{\{e_i,f\}}(s,t)|\}$$

As in the algorithm DSERP, we start from the head, t, of $\mathcal{P}_{G_{-i}}(s,t)$ to compute the replacement paths for the pairs $\{e_i, f\} \forall f \in F_i^t$. For the pair $\{e_i, e_j\}$ with $e_j = (u_j, u_{j+1})$ the $|\mathcal{R}^v_{\{e_i,e_j\}}(s,t)|$ values are updated as follows:

$$|\mathcal{R}^v_{\{e_i,e_j\}}(s,t)| = MIN\{|\mathcal{R}^v_{e_i,e_{j+1}}(s,t)|, d_{G_{-i}}(s,v) + d_{G_{-\mathcal{P}(s,t)}}(v,u_{j+1}) + d_{G_{-i}}(u_{j+1},t)\} \quad (5)$$

Also, we do not need to perform another APSP computation. The APSP computation performed in the initial step provides enough information since we only need the candidate replacement path from v to some node $u \in \mathcal{P}_G(s,t)$ which does not use the edges e_i and (u_j, u_{j+1}).

We formally present our algorithm EPRP below. The algorithm takes as input a directed weighted graph $G(V,E)$, and two specified nodes $s, t \in V$. $\mathcal{P}_G(s,t)$ can be computed in $O(m + n \log n)$ time if not supplied as part of the input. For simplicity, we assume $\mathcal{P}_G(s,t)$ to be part of input.

Algorithm EPRP

- Input: $G(V,E)$, two special nodes $s, t \in V$, and the shortest path from s to t in G, $\mathcal{P}_G(s,t) = \{e_1, e_2, \ldots, e_p\} = \{s = u_0, u_1, \ldots, u_p = t\}$.
- Initialization: Perform an APSP computation on the graph $G \backslash E(\mathcal{P}_G(s,t))$. Also, we compute the shortest paths tree of s in each of the p graphs $G \backslash e_i$ for $1 \leq i \leq p$.
- For each edge $e_i \in \mathcal{P}_G(s,t)$, do:
  - Construct the shortest paths tree $\mathcal{T}_s^i$ of s in the graph $G_{-i} = G \backslash e_i$. Let $\mathcal{P}_{G_{-i}}(s,t)$ denote the path from s to t in $\mathcal{T}_s^i$ (and G_{-i}).
  - List out the edges common in $\mathcal{P}_{G_{-i}}(s,t)$ and $\mathcal{P}_G(s,t)$ as $F_i = \mathcal{P}_G(s,t) \cap \mathcal{P}_{G_{-i}}(s,t) = F_i^s \cup F_i^t$
  - Solve the instance of the replacement paths problem only for the edges in F_i^t as described above.

The $O(n^3)$ time complexity comes from the fact that each instance of the replacement paths problem we solve requires $O(n^2)$ time. Over the entire course of the algorithm, we invoke the replacement paths subroutine p times, where $p = |F|$. Since p can be as high as $O(n)$, the algorithm takes $O(n^3)$ time. The APSP computation is required only once in the beginning of the algorithm and can be performed using any standard algorithm: Dijkstra's algorithm takes $O(mn + n^2 \log n)$ time and Floyd's algorithm takes $O(n^3)$ time. Either can be used since it does not affect the asymptotic time complexity of our algorithm.

We have thus established the following theorem about the Edge-Pairs-Replacement-Paths problem.

Theorem 4.1 *Every instance of the Edge-Pairs-Replacement-Paths problem can be solved in $O(n^3)$ time. If $m = \Theta(n^2)$, this bound is tight for path-comparison based algorithms as shown by the $\Omega(mn)$ lower bound for the problem.*

Proof: The correctness of the algorithm EPRP and the claim on the time bound follow from the discussion on the algorithm EPRP. □

Corollary 4.1 *Given a directed weighted graph $G(V,E)$, the two most vital arcs of a given s-t shortest path $\mathcal{P}_G(s,t)$, defined as the two edges lying on the path whose deletion*

produces the largest increase in the s-t shortest path distance, can be found in $O(n^3)$ time.

Proof: Associate with the pair $\{e_i, e_j\}$, where $e_i, e_j \in \mathcal{P}_G(s,t)$, $\mathcal{D}(e_i,e_j) = |\mathcal{P}_{G\setminus\{e_i,e_j\}}(s,t)| - |\mathcal{P}_G(s,t)|$. The two most vital arcs are $\{e,f\}$ such that $\mathcal{D}(e,f) = MAX_{e_i,e_j}\{\mathcal{D}(e_i,e_j)\}$. After solving the EPRP instance, the remaining steps can be performed in constant time per pair $\{e_i,e_j\}$, taking an additional total time of $O(n^2)$. □

5 Concluding Remarks

In this paper we have discussed some problems related to computing shortest paths in networks susceptible to link failures. The directed version of the replacement paths problem is especially interesting since it finds applications in several network problems as well as those related to algorithmic mechanism design[12]. Although our $O(T_{apsp}(m,n))$ replacement paths algorithm is an improvement over the naive algorithm for many practical cases, there is a significant gap between the lower bound of $\Omega(min(n^2, m\sqrt{n}))$ presented in [9] and our new upper bound. Note that our subcubic $DSERP$ algorithm is not a path comparison based algorithm since it uses APSP algorithms based on fast-matrix-multiplication techniques, whereas the lower bound of [9] holds only for path-comparison based algorithms. Another point to be noted for the $EPRP$ lower bound presented in this paper is that the construction employs an input graph which has $\Theta(n)$ edges on the s-t shortest path. Using the same techniques, problem instances can be constructed which have only $O(r)$ edges on the s-t shortest path and impose a (weaker) lower bound of $\Omega(mr)$ on the $EPRP$ problem for path-comparison based algorithms.

Another interesting open problem is to solve the directed EPRP instances for when *any* pair of edges from the entire graph can fail, rather than only the edges lying on the given s-t shortest path.

The EPRP algorithm can be generalized to compute the s-t shortest path when not just two, but k edges of the original shortest path fail, for all such combinations of edges lying on the s-t shortest path. The time complexity of this algorithm would be $O(n^{k+1})$, instead of $O(n^k(m + n\log n))$ required by the naive algorithm based on recomputation. The algorithm goes along the lines of the EPRP algorithm: After an initial APSP computation on $G\setminus\mathcal{P}_G(s,t)$, for each set $E^{k-1} \subseteq E$ of $k-1$ edges of $\mathcal{P}_G(s,t)$, first construct the shortest paths tree $\mathcal{T}_s^{k-1}$ of s in the graph $G_{-E^{k-1}} = G\setminus E^{k-1}$. Find the subpath F common to $\mathcal{P}_{G_{-E^{k-1}}}(s,t)$ and $\mathcal{P}_G(s,t)$ that ends at t, and find the s-t replacement paths for the k edges $E^{k-1}\cup f\ \forall f \in F$. Further, equations similar to (4), (5) can be established and used in a similar way for the k-Edges Replacement Paths problem. We omit further details from this abstract.

Acknowledgements

The first author thanks Subhash Suri for important discussions during the preliminary stages of this work.

References

[1] 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.

[2] H. Choi, S. Subramaniam, and H.-Ah. Choi. On double-link failure recovery in WDM optical networks. In *IEEE INFOCOM*, 2002.

[3] E.W. Dijkstra. A note on two problems in connection with graphs. In *Numerische Mathematik*, pages 1:269-271, 1959.

[4] M.L. Fredman. New bounds on the complexity of the shortest path problem. In *SIAM J. of Computing*, pages 5:83-89, 1976.

[5] 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.

[6] Roch Gurin and Ariel Orda. Computing shortest paths for any number of hops. *IEEE/ACM Transactions on Networking (TON)*, 10(5):613–620, 2002.

[7] J. E. Hershberger and S. Suri. Erratum to "vickrey pricing and shortest paths: What is an edge worth?". In *IEEE Symp. Found. of Comp. Sci.* [8], pages 809–809.

[8] J. E. Hershberger and S. Suri. Vickrey prices and shortest paths: What is an edge worth? In *IEEE Symp. Found. of Comp. Sci.*, pages 252-259, 2001.

[9] J. E. Hershberger, S. Suri, and A. M. Bhosle. On the difficulty of some shortest paths problems. In *Proceedings of STACS '03*, 2003.

[10] D.R. Karger, D. Koller, and S.J. Phillips. Finding the hidden path: Time bounds for all-pairs shortest paths. In *IEEE Symp. Found. Comp. Sci.*, pages 560-568, 1991.

[11] 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.

[12] N. Nisan and A. Ronen. Algorithmic mechanism design. In *Proc. 31st Annu. ACM Sym. Theory of Comput.*, pages 129-140, 1999.

[13] 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.

[14] T. Takaoka. A new upperbound on the complexity of the all pairs shortest path problem. In *Information Processing Letters 43*, pages 195-199, 1992.

[15] U. Zwick. All pairs shortest paths in weighted directed graphs exact and almost exact algorithms. In *IEEE Symp. Found. of Comp. Sci.*, pages 310-319, 1998.