Faster Replacement Paths Algorithm for Undirected, Positive Integer Weighted Graphs with Small Diameter

Jay Mahadeokar and Sanjeev Saxena

Dept. of Computer Science and Engineering,
Indian Institute of Technology,
Kanpur, India-208 016

Abstract. We consider the replacement path problem for undirected graphs in case of edge failures. Given a 2-edge connected graph $G(V, E)$, where $n = |V|$ and $m = |E|$, for each edge e on the shortest $s-t$ path of G, we are to report the shortest $s-t$ path in $G \setminus e$. If d is the diameter of the graph, the proposed algorithm takes $O(m + d^2)$ time.

For graphs where $d = O(\sqrt{m})$, typically dense graphs, or graphs with small diameter we have a linear time solution.

1 Introduction

The replacement paths problem in case of edge failure is:

> given a 2-edge connected graph $G(V, E)$, for each edge e on the shortest $s-t$ path of G, report the shortest $s-t$ path in $G \setminus e$.

There exists a trivial solution where replacement path for every edge e is computed by running shortest path algorithm independently on $G \setminus e$. Let d be the diameter of the graph. As there can be at most d edges on the shortest path, this takes $O(d(m + n \log n))$ time.

Malik et.al.[6] describe an $O(m + n \log n)$ time solution for the problem on undirected graphs with positive weights. This algorithm was rediscovered by Hershberger and Suri [3] (see e.g. [7,4] for brief history). Their algorithm consists of two main parts:

- Finding the shortest path trees rooted at s and t.
- Reporting the replacement paths for each edge on $s-t$ path using the shortest path trees.

In this paper we describe a new algorithm for the second part. The proposed algorithm takes $O(m + d^2)$ time. If $d = O(\sqrt{m})$ this results in a linear time algorithm for the second part. If the graph has integer weights, then we use the linear time algorithm of Thorup [11] for the first part. Or, if the graph is planar, then we can use the $O(n)$ time shortest path algorithm described by Henzinger et al.[5] instead, to get a linear time algorithm for the problem.

Nardelli et al [8] have described an $O(m\alpha(m,n))$ time algorithm for solving the most vital edge problem, which also solves the replacement paths problem. They also use the linear time algorithm of Thorup [11] for the first part. For the second part, they use the transmuter [9,10] data structure described by Tarjan. Note that the technique described in this paper is simpler and easier to implement.

2 Replacement Paths in Case of Edge Failure

We assume that the graph $G(V,E)$ is undirected and edges have positive integer weights. For $s,t \in V$, we will denote the shortest path from s to t by $P = \{v_0, v_1, v_2 ... v_t\}$ with $s = v_0$ and $t = v_t$. So, for every edge $e = (v_k, v_{k+1})$, $k = 0...t-1$, we want to report the corresponding replacement path, that is the shortest $s-t$ path in the graph $G \setminus e$.

Like[6,3] we also find the shortest path trees X rooted at s and Y rooted at t respectively. Any edge $e = (x,y) \in P$ divides the tree X into two disjoint trees (X_s) and (X_t) such that $s \in X_s$ and $t \in X_t$.

Let the "reduced cost" of an edge (u,v) be

$$\varphi(u,v) = d(s,u) + c(u,v) + d(v,t)$$

Here $d(s,u)$ is the length of the shortest path from s to u, $c(u,v)$ is the cost of edge (u,v) and $d(v,t)$ is the length of the shortest path from v to t. Since G is an undirected graph, $d(v,t) = d(t,v)$, the length of the shortest path of a node v from t in the shortest path tree Y. Hershberger and Suri [3], use the property that the shortest path in the graph $G \setminus e$ is the path through a non-tree edge (u,v) such that $u \in X_s$, $v \in X_t$ and $\varphi(u,v)$ is minimum.

We say that such edge (u,v) is the *replacement edge* of e and $\varphi(u,v)$ is the *replacement cost* of e.

Note that the edge (u,v) belongs to the fundamental cut set [2] of edge e.

We preprocess non-tree edges in $O(m + d^2)$ time, so that for any given tree edge (v_k, v_{k+1}) on P, the corresponding replacement edge (u,v) can be found in $O(d)$ time. Since there can be $O(d)$ edges in P, we can compute the replacement cost for each edge in $O(d^2)$ time.

Let us carry out preorder traversal on the shortest path tree X. Let $pre(v)$ denote the preorder number of node v and $desc(v)$ denote the number of descendants of v, including v. Let

$$\alpha(v) = pre(v) + desc(v)$$

Let T_i be the set of nodes that are descendents of node v_i (including v_i) but not of v_{i+1}. Thus, the nodes of the tree are partitioned into sets $T_0, T_1 ... T_{t-1}$.

Let N_i be the set of non-tree edges, between T_i and T_{i+b} (for $b \geq 1$). Less informally:

$$N_i = \{(x,y) \mid (x \in T_i \text{ and } y \in T_j) \; \forall \; j \geq i+1\}$$

Thus, the non-tree edges are partitioned into sets $N_0, N_1, \ldots, N_{t-1}$.

2.1 Preprocessing

For graphs with integer weights, the shortest path trees can be obtained in $O(n+m)$ time using algorithm described by Thorup[11]. We can easily determine the preorder number and number of descendents of nodes in linear time. Hence the values $pre(v)$, $desc(v)$, $\alpha(v)$ and $\varphi(u,v)$ are determined in $O(n)$ time.

We use bucket sort to sort these non-tree edges (say (u,v)) on the $pre(v)$, the preorder number of the second entry v, in $O(n+m)$ time.

By post order traversal, we can easily determine the sets of tree edges $T_0, T_1, \ldots, T_{t-1}$ in linear time. Then by looking at each non-tree edge one by one, we can put it the sets N_j, in constant time. Thus, the sets $N_1, \ldots, N_{t-1}$ of non-trees edges can also be constructed in $O(m+n)$ time.

Berkman et al. [1] has shown that an array $A[1:n]$ can be preprocessed in linear time to answer range minima queries of the following form in constant time:

For $1 \leq i, j \leq n$ find the smallest item in $A[i], A[i+1] ..., A[j-1], A[j]$.

We preprocess each set N_i independently to answer range minima query with reduced cost φ as the key. Since the total number of elements in all the sets together is $O(m)$ we need $O(m)$ time to perform this step.

For each set N_i and for every vertex v_j, $j \geq i+1$ on P, we maintain the rank of $pre(v_j)$ in N_i. In other words we find the "pointer" $Pre[i,j]$ which will points to the smallest element in N_i which is greater than or equal to $pre(v_j)$.

These pointers can be obtained by doing a binary search for each $pre(v_j)$ in N_i, but that will take $(t \log |N_i|)$ time.

Alternatively, we can also get the ranks by merging sorted array $pre(v_i), pre(v_{i+1}), \ldots, pre(v_t)$ with N_i. This will take $O(t+|N_i|)$ time. Or total time for all i's will be $\sum O(t+|N_i|) = O\left(\sum t + \sum |N_i|\right) = O(t^2+m)$.

We similarly find pointer $A[i,j]$ which points to largest element in N_i which is smaller than or equal to $\alpha(v_j)$.

Thus the total time taken by algorithm preprocess is $O(m+d^2)$

2.2 Reporting Replacement Paths

To report the replacement path for edge (v_k, v_{k+1}) we are interested in all (non-tree) edges (u,v) which connect a non-descendant of v_{k+1} to a descendant of v_{k+1}. Because of preprocessing, we can assume that for any non tree edge (u,v), $pre(u) < pre(v)$

Case 1: (v is not a descendant of v_{k+1}) Let us first consider the case when u is a descendant of v_{k+1} and v is not.

As u is a descendant:

$$pre(v_{k+1}) \leq pre(u) < \alpha(v_{k+1})$$

and as v is a non-descendant:

$$\alpha(v_{k+1}) \leq pre(v) \leq n$$

As u is a descendant of v_{k+1}, u will be in $T_{k+1} \bigcup \ldots \bigcup T_t$ and as v is a non descendant, v will be in $T_0 \bigcup T_1 \bigcup \ldots \bigcup T_k$. As the edges in N_i have one point in T_i, all these edges will be present in $N_0 \bigcup N_1 \bigcup \ldots \bigcup N_k$. Hence, we only need to look at edges in $N_0 \bigcup N_1 \bigcup \ldots \bigcup N_k$.

Case 2: (u is a non-descendant) In the other case, u is not a descendant of v_{k+1} but v is a descendant of v_{k+1}.
As u is not a descendant:

$$0 < pre(u) < pre(v_{k+1})$$

and as v is a descendant of v_{k+1},

$$pre(v_{k+1}) \leq pre(v) < \alpha(v_{k+1})$$

Further, u will be in $T_0 \bigcup T_1 \bigcup \ldots \bigcup T_k$ and v will be in $T_{k+1} \bigcup \ldots \bigcup T_t$. As edges in N_i have one point in T_i, all these edges will be present in $N_0 \bigcup N_1 \bigcup \ldots \bigcup N_k$. Hence, we only need look at edges in $N_0 \bigcup N_1 \bigcup \ldots \bigcup N_k$.

As the process of finding the edge in the two cases is similar, we will only discuss implementation of the first case.

Let $C_k = N_0 \cup N_1 \cup \ldots \cup N_k$ be the set of candidate edges which satisfy the conditions of Case 1. From this set we want to report the edge with the least reduced cost φ.

Because the way we constructed N_i during preprocessing, we can assume that edges in each N_i are sorted according according to $pre(v)$.

For each set T_i for $i \leq k$, we find the edge with minimum φ value between $pre(v_{k+1})$ and $\alpha(v_{k+1})$ by performing the following range minima query:

$$(N_i,\ Pre[i, k+1],\ A[i, k+1]).$$

These are edges in N_i with $pre(v)$ between $pre(v_{k+1})$ and $\alpha(v_{k+1})$; in other words we are only looking at those edges of N_i for which v is a descendant of v_{k+1}.

Each of these queries takes $O(1)$ time. As there are k queries, total time will be $O(k)$. But, since $k = O(d)$ time is $O(d)$.

The replacement edge in Case 1, for (v_k, v_{k+1}) is the edge corresponding to the minimum of these range minima queries.

By a similar procedure, we can find a replacement edge in the other case for (v_k, v_{k+1}). The required replacement edge will be the one with smaller φ-value.

We repeat the procedure, for each tree edge $(v_k, v_{k+1}) \in P$. The total time to find all these replacement edges is $O(dk) = O(d^2)$.

If (u, v) is the replacement edge with minimum reduced cost $\varphi(u, v)$ among all these replacement edges, then the replacement path R for P as the path $(P(s, v) \in X)$ $+(u, v) + (P(v, t) \in Y)$.

Thus the total time required to find the replacement path for P including the preprocessing step is $O(m + d^2)$.

3 Conclusion

We have proposed an $O(m+d^2)$ time algorithm for the replacement paths problem in case of edge failures for undirected graphs with positive integer weights. If diameter d of the graph is $O(\sqrt{m})$ then, our algorithm runs in linear time. For undirected, integer weighted dense graphs or sparse graphs with small diameter, our algorithm performs better than the existing algorithms. For planar graphs, our algorithm takes $O(n+d^2)$ time. Thus we conclude that the replacement path problem can be solved as efficiently as the shortest path problem, if diameter of graph is $O(\sqrt{m})$

References

1. Berkman, O., Schieber, B., Vishkin, U.: Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values. J. Algorithms 14, 344–370 (1993)
2. Deo, N.: Graph Theory with Applications to Engineering and Computer Science. Prentice-Hall (1974)
3. Hershberger, J., Suri, S.: Vickrey prices and shortest paths: what is an edge worth? In: Proc. FOCS, pp. 252–259 (2001)
4. Hershberger, J., Suri, S.: Erratum to ”Vickrey Pricing and Shortest Paths: What is an Edge Worth? In: FOCS, p. 809 (2002)
5. Henzinger, M.R., Klein, P., Rao, D., Subramanian, S.: Faster shortest-path algorithms for planar graphs. J. Comput. Syst. Sci. 55, 3–33 (1997)
6. Malik, K., Mittal, A.K., Gupta, S.K.: The most vital arcs in the shortest path problem. Oper. Res. Letters 8, 223–227 (1989)
7. Nardelli, E., Proietti, G., Widmayer, P.: Finding the most vital node of a shortest path. Theoretical Computer Science 296, 167–177 (2003)
8. Nardelli, E., Proietti, G., Widmayer, P.: A faster computation of the most vital edge of a shortest path. Inf. Process. Lett. 79(2), 81–85 (2001)
9. Tarjan, R.: Sensitivity Analysis of Minimum Spanning Trees and Shortest Path Trees. Information Processing Letters 14(1), 30–33 (1982)
10. Tarjan, R.: Applications of path compression on balanced trees. J. ACM 26, 690–715 (1979)
11. Thorup, M.: Floats, Integers, and Single Source Shortest Paths. In: Meinel, C., Morvan, M. (eds.) STACS 1998. LNCS, vol. 1373, pp. 14–24. Springer, Heidelberg (1998)