An Improved Algorithm for Finding the Anti-block Vital Edge of a Shortest Path

Zhe Nie Yueping Li, Member IAENG*

Abstract—**This paper presents an improved algorithm to find the anti-block vital edge of a shortest path. We release the constraint that there is only one shortest path between two nodes introduced by Su, Xu and Xiao. We use the technique developed by Nardelli, Proietti and Widmayer and give a improvement in search strategy. Our algorithm runs in $O(m + n \log n)$ time which is superior to the previous one whose complexity is in $O(mn)$, where n and m denote the number of nodes and edges in the graph. In addition, our algorithm can be further improved to run in $O(m\alpha(m, n))$ time, where α is the functional inverse of the Ackermann function.**

Keywords: ***graph algorithms, shortest path, detour, critical edge***

1 Introduction and Preliminary

The survivability of communication network is a critical issue. It has been studied intensively. We focus on a particular type of it: How is a network affected by an edge (link) failure? Our scenario assumes that we route along a shortest path in the network from the source to the destination. When an edge fails, we need to choose another path, which probably is a shortest path does not contain the failed edge. This problem is called *path replacement* in literature [1]. From the management point of view, it is valuable to evaluate the effect by the failure of a link. The classical problem is to find a *most vital edge* (MVE): the edge whose removal results in the largest increase of the length with respect to a shortest path. Corley and Sha [3] proposed an $O(mn + n^2 \log n)$ time algorithm to solve this problem. And a more effective algorithm was developed by Malik, Mittal and Gupta [5] which runs in $O(m + n \log n)$ time.

Most previous works paid attention to the length of the replacement path minus the length of the shortest path. For example, the shortest path from s to t is $P_G(s, t) = s, d, b, g, t$ which is illustrated with wavy lines. We denote the length of path P by $|P|$. So we have $|P_G(s, t)| = 8$. It is easy to verify that the edge (s, d) is the most vital edge. We have $P_{G-(s,d)}(s, t)$ whose length is 16. Thus, the edge (s, d) is critical. However, Su, Xu and Xiao [8] proposed a different parameter for measuring the vitality of an edge of a shortest path. They focused on an edge $e = (u, v)$ in $P_G(s, t)$ whose removal produces a replacement path at vertex u such that $|P_{G-e}(u, t)|/|P_G(u, t)|$ is maximum. They defined such an edge as the anti-block vital edge (AVE for abbreviation).

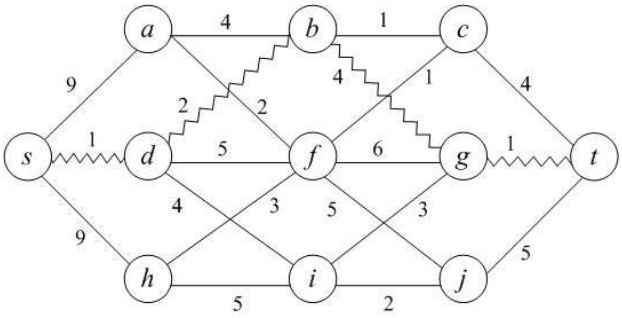

Figure 1. A shortest path.

Our scenario is that a traveller may reach a vertex u but the edge (u, v) which is intended to pass fails suddenly. The traveller will route from u to t through a shortest path $P_{G-(u,v)}(u, t)$. It is natural that the maximum ratio is one of the key parameters for measuring a route strategy. For instance, if the edge (s, d) is failure, $|P_{G-(s,d)}(s, t)|/|P_G(s, t)| = 2$. However, if the edge (g, t) is not available, $|P_{G-(g,t)}(g, t)|/|P_G(g, t)| = 9$. It implies that the edge (g, t) is more important than the edge (s, d) from this measure of view. Note that the increase length of $|P_{G-(s,d)}(s, t)| - |P_G(s, t)|$ equals $|P_{G-(g,t)}(g, t)| - |P_G(g, t)|$ whose value is 8. Hence, the edges (s, d) and (g, t) have no difference with respect to the increase of the length of a shortest path.

This paper is organized as follows: In Section 2, we define the problem formally. In Section 3, our algorithm is presented. In Section 4, an improvement is given. Experimental results and analysis are proposed in Section 5 and 6, respectively. Finally, the conclusions and future works are given in Section 7.

2 Definition and Terminology

Let $G = (V, E)$ be a simple graph with vertex-set $V(G)$ and edge-set $E(G)$ where $|V(G)| = n$ and $|E(G)| = m$.

*Zhe Nie: Shenzhen Polytechnic, Xili, Shenzhen P.R. China 518055 Email:niezhe@oa.szpt.net; Yueping Li: Sun Yat-sen University, Department of Computer Science, Guangzhou P.R. China 510275, Email: leeyueping@gmail.com

Let $w(e)$ be a positive real length for each edge $e \in E(G)$. A graph $H = (V(H), E(H))$ is called a subgraph of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. And if $V(H) = V(G)$, H is said to be a spanning subgraph of G.

A connected acyclic spanning subgraph of G is called a spanning tree of G. A single-source shortest paths tree (SPT) $S_G(r)$ is a spanning tree of G rooted at r and consisting of the union of the shortest paths. It is straightforward that there is exactly one shortest path from r to v for each $v \in V(G \setminus r)$.

A graph G is connected if there exists a path from u to v for any two vertices $u, v \in V(G)$. We call a graph G 2-edge-connected if $G - e$ is connected for any edge $e \in E(G)$. We consider undirected 2-edge-connected graphs in this paper.

Let $P_{G-e}(s,t)$ be a shortest path between s and t. As mentioned above, we call $P_{G-e}(s,t)$ a replacement shortest path for v. Denote its length by $d_{G-e}(s,t)$.

The anti-block coefficient of an edge $e = (u,v) \in P_G(s,t)$ is the ratio $c_{u,t}$ of $d_{G-e}(u,t)$ to $d_G(u,t)$.

The anti-block vital edge (AVE) with respect to $P_G(s,t)$ is the edge $e' = (u',v') \in P_G(s,t)$ whose removal results in $c_{u',t'} \geq c_{u,t}$ for any edge $e = (u,v)$ of $P_G(s,t)$.

3 Description of the Algorithm

Su, Xu and Xiao [8] proposed an $O(mn)$ time to find the anti-block edge with respect to a shortest path. But they assumed that there is only one shortest path between the source s and the destination t, which has limited application. We eliminate this constraint and develop a more effective algorithm.

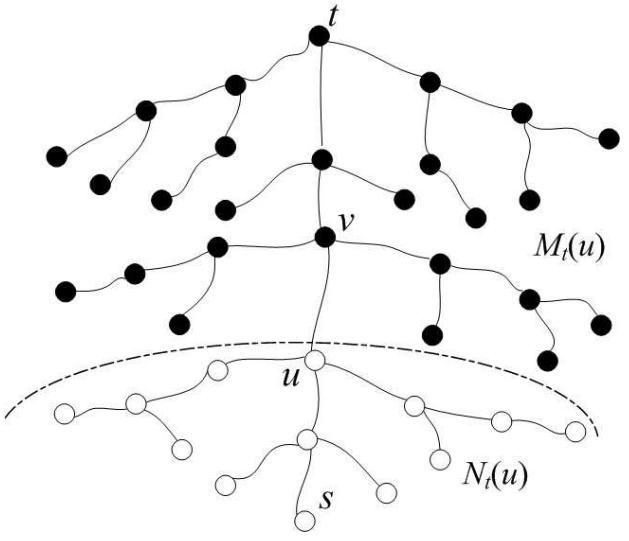

Figure 2. $S_G(t)$.

At first, we compute the shortest path tree $S_G(t)$ rooted at t. It is natural to develop an algorithm by building $S_{G-e}(t)$ for each $e \in P_G(s,t)$ which runs in $O(mn)$ time. The reader can refer to Su, Xu and Xiao [8]. However, it is too expensive. In the light of [7], we adopt the Fibonacci heaps [4] in order to improve the algorithm to run in $O(m + n\log n)$ time.

Let $e = (u,v)$ be an edge of $P_G(s,t)$ where the vertex u is closer to s than v. Let $M_t(u)$ denote the set of vertices reachable in $S_G(t)$ from t avoiding passing the edge (u,v). Let $N_t(u) = V(G) - M_t(u)$. An example of $M_t(u)$ and $N_t(u)$ is illustrated in Fig. 2. It is straightforward that for any vertex x in $M_t(u)$, we have $d_{G-e}(x,t) = d_G(x,t)$.

We define the edges between the partition $N_t(u)$ and $M_t(u)$ as follows:
$E_t(u) = \{(x,y) \in E(G) - (u,v) | x \in N_t(u)$ and $y \in M_t(u)\}$.

Suppose the traveller has arrived at the vertex u, but the edge $e = (u,v)$ fails at that time. Then the traveller has to route avoiding the edge e. That is, the detour $P_{G-e}(u,t)$ must contain an edge in $E_t(u)$ of which an example is shown in Fig. 3. It implies that the length of detour satisfies the following formula:

$$d_{G-e}(u,t) = \min_{(x,y)\in E_t(u)} \{d_{G-e}(u,y) + w(x,y) + d_{G-e}(x,t)\} \quad (1)$$

.

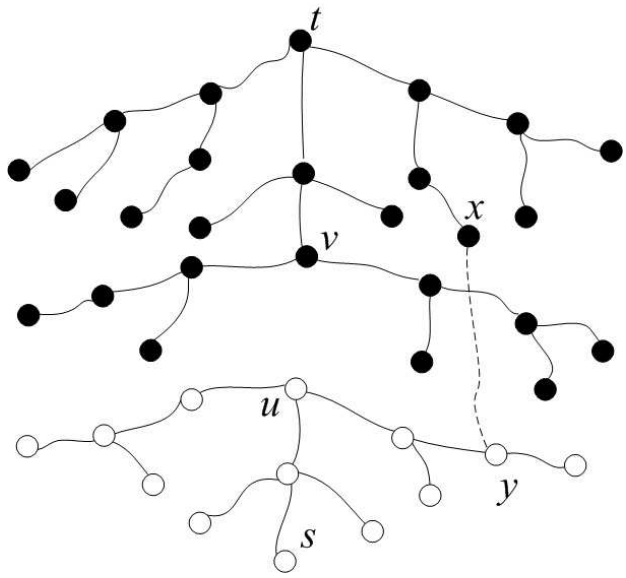

Figure 3. A detour $P_{G-(u,v)}(u,t)$.

From the structure of $S_G(t)$, it implies that $d_{G-e}(u,y) = d_G(y,t) - d_G(u,t)$. Note that $d_{G-e}(x,t) = d_G(x,t)$. Thus, Formula (1) can be computed in $O(1)$ time. Since we can check each edge in $E_t(u)$ in $O(m)$ time, it is straightforward that our problem can be solved in $O(mn)$ time as mentioned above. We now propose a refined algorithm by means of the technique developed by Nardelli, Proietti

and Widmayer [7]. Suppose the path $P_G(s,t)$ is $s_0(=s)$, s_1, ..., s_{r-1}, $s_r(=t)$ and $e_i = (s_i, s_{i+1})$.

For each e_i, we adopt a Fibonacci heap to build a priority queue whose element stores the shortest path $P_{G-e_i}(s_i,t)$. In the light of Formula (1), we use $Q_i(y)$ to record the shortest path passing through the vertex y and avoiding the edge e_i. We define

$$Q_i(y) = \min_{(x,y)\in E_t(s_i)} \{d_G(t,x) + w(x,y) + d_G(y,s_i)\} \quad (2)$$

It is clear that we need to maintain the queue Q_i for the vertices of $N_t(s_i)$ only. Note that $N_t(s_i) \subset N_t(s_{i+1})$. Hence, we set $i = r$ and calculate $Q_r(y)$ at the first step. Since $i = r$, the edge e_i is empty. Thus, $Q_r(y)$ just records the value of $d_G(y,t)$. Once the queue $Q_i(y)$ is calculated out, we continue to compute $Q_{i-1}(y)$ where $i > 0$.

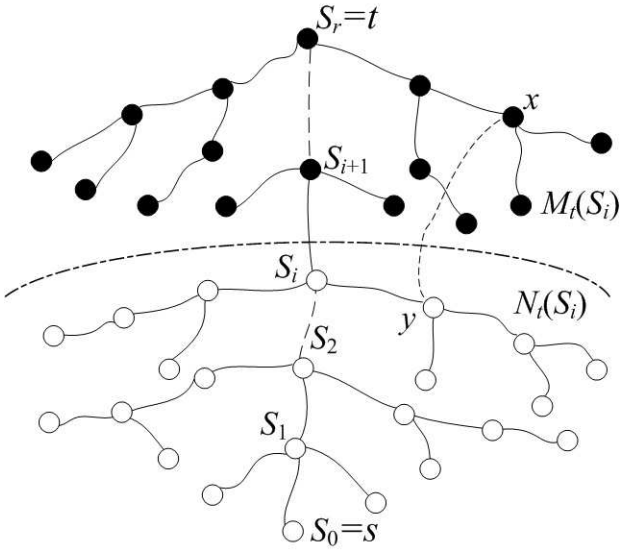

Figure 4. S_i and S_{i+1}.

Nardelli et al. [7] pointed out if we use Formula (2) as a key of the priority queue, the cost will be expensive, since when the next edge e_{i-1} is considered, we have to decrease the value by $w(e_{i-1})$ for all the elements in the queue. Thus, they gave the appropriate key as follows:

$$K_i(y) = \min_{(x,y)\in E_t(s_i)} \{d_G(t,x) + w(x,y) + d_G(y,s)\} \quad (3)$$

It is clear that if the vertex y remains in $N_t(s_{i-1})$, the value $K_i(y) - d_G(t,s_{i-1})$ still records the length of a candidate path. We now give the procedure of the algorithm introduced by Nardelli et al. [7].

Nardelli's Algorithm
Input: A graph G with a shortest path $P_G(s,t)$
Output: d_{G-e} for any $e \in E(P_G(s,t))$
(1) Build a shortest path tree rooted at t,
denoted by $S_G(t)$
(2) Let $K(y) = d_G(s,t)$ for all $y \in V(G)$
and build a Fibonacci heap using $K(y)$ as key
(3) Suppose $P_G(s,t) = s_0(=s)$, s_1, ..., s_{r-1}, $s_r(=t)$
(4) Let $i = r$ and $e_i = (s_i, s_{i+1})$
(5) while $i > 0$ do
(6) Begin
(7) Let S be the set of $N_t(s_i) - N_t(s_{i-1})$
(8) Remove the key $K(x)$ from the heap if $x \in S$
(9) For each $x \in S$, search its neighbors:
If there is an edge (x,y) where $x \in N_t(s_{i-1})$,
we compute $d_G(t,y) + w(y,x) + d_G(x,t)$;
If the value is less than $K(y)$, we assign it to $K(y)$
(10) Let c be the minimum key of the heap
(11) Let $d_G(s_{i-1},t) = c - d_G(t,s_{i-1})$
(12) Decrease i by 1
(13) End(while)

Since the values of d_{G-e} for any $e \in E(P_G(s,t))$ have been obtained, it is easy to calculate the maximum anti-block coefficient along the edges of $P_G(s,t)$. That is, we can determine the anti-block edge in this way.

4 Improvement

In this section, we propose an improvement of the search strategy and discuss the time-space trade-off. The fastest algorithm to perform Step (1) introduced by Fredman and Tarjan [4] adopts the adjacent list to store the graph. It runs in $O(m + n\log n)$ time.

It is well known that the adjacent list can be implemented by means of link list. Furthermore, we add two links between the two instances which stand for the same edge in graph which is shown in Fig. 5. Furthermore, we maintain the array which stores the tails of the link list of certain vertices.

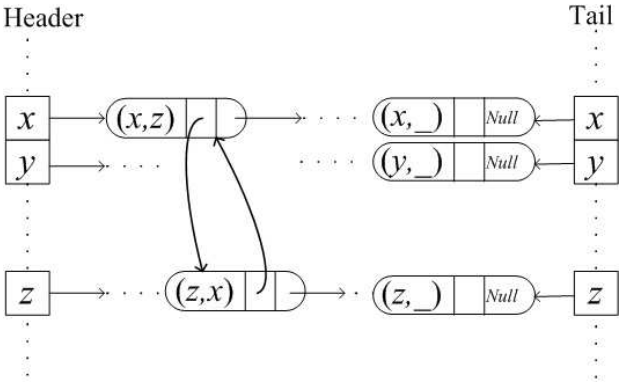

Figure 5. The structure of adjacent list.

We change Step (9) as follows:
(9) For each $x \in S$, search its edge link list from the header:

(9.1) Let e be the first edge in the list.
(9.2) while $e <> Null$ do
(9.3) Begin if e is marked, then break;
(9.4) Suppose $e = (x, y)$;
(9.5) If $x \in N_t(s_{i-1})$, we compute $d_G(t, y) + w(y, x) + d_G(x, t)$;
(9.6) If the value is less than $K(y)$, we assign it to $K(y)$;
(9.7) Mark the edge e;
(9.8) Move e to the tail of the link list of the vertex y;

We use more n units of space to store the tail of the link list of each vertex. But we avoid to search one edge twice. Thus, the improved algorithm diminishes the complexity by m units of time. Since m is in $O(n^2)$ usually, our improvement makes sense.

Our improvement above focuses on the time-space tradeoff. We now propose a method to perform Step (8) effectively and then present an alternative method which uses less space but takes more time. As known, it is expensive to locate the node which is associated with the vertex y in Fibonacci heap since the key is $K(y)$ not y. Hence, we also use an array to record the node in the heap for each vertex in the graph. But if we prefer space to time, we does not record the position of the corresponding node for each vertex. We eliminate Step (8) and change Step (10) as follows:
(10.1) Let c be the minimum key of the heap;
(10.2) If c is the value of $K(y)$ and the vertex y is not in $N_t(s_{i-1})$, then delete the key c and Goto Step (10.1);

It implies the alternative algorithm spares n units of space but takes more time for the operations of the Fibonacci heap. However, the time complexity remains in $O(m + n \log n)$.

5 Experimental Results

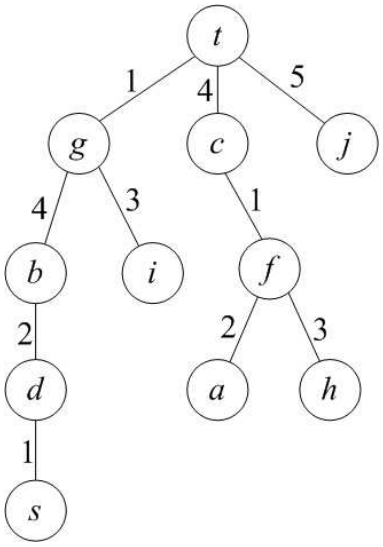

Figure 6. The shortest path tree.

The shortest path tree of the graph in Fig. 1 is shown in Fig. 6. The optimal detours are shown in Fig. 7 for each vertex when the edge from the vertex to its parent fails. The first edges of the detours are marked with dotted lines.

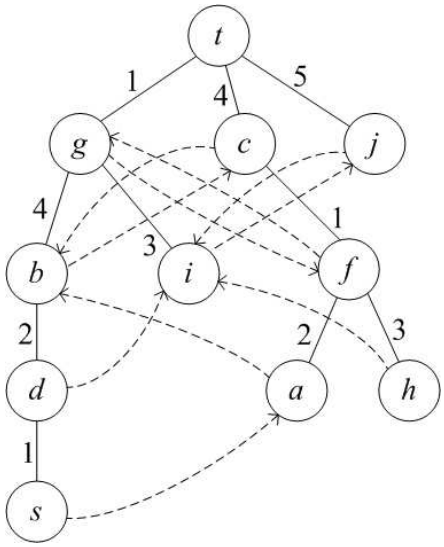

Figure 7. The optimal detours.

And the following table presents the lengths of the optimal detours.

Table 1: $P_G(x,t)$, $d_{G-(x,y)}(x,t)$ and $c_{x,t}$

Edge(x,y)	$P_G(x,t)$	$d_{G-(x,y)}(x,t)$	$c_{x,t}$
(s,d)	8	16	2
(d,b)	7	8	8/7
(b,g)	5	5	1
(g,t)	1	9	9
(i,g)	4	7	7/4
(a,f)	7	9	9/7
(h,f)	8	9	9/8
(f,c)	5	7	7/5
(c,t)	4	6	3/2
(j,t)	5	6	6/5
(b,c)	5	5	1
(c,t)	4	6	3/2

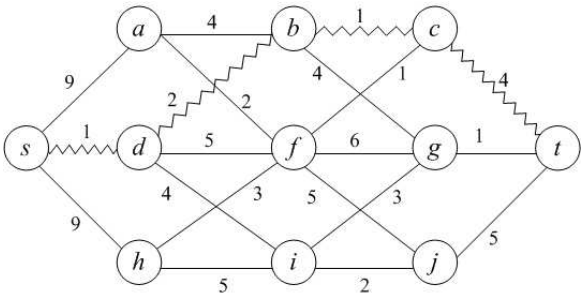

Figure 8. An alternative shortest path from s to t.

As mentioned in Section 1, though $d_{G-(g,t)} = d_{G-(s,d)}$, the anti-block coefficient of the edge (g,t) is much larger than the one of (s,d). So the edge (g,t) is critical in the path. From this point of view, we had better choose the shortest path to be $P = s,d,b,c,t$. The maximum anti-block coefficient of P is 2 since the $c_{b,t} = 1$ and $c_{c,t} = 3/2$ with respect to the edges (b,c) and (c,t), respectively. Thus, the route of P is better than the route s,d,b,g,t which is shown in Fig. 1.

6 Analysis of the Algorithm

According to Nardelli's algorithm, it can be concluded that it has no limitation on the number of the shortest path from the source to the destination. Thus, so does our algorithm. It is clear that our algorithm runs in $O(m + n\log n)$ time and $O(m)$ space.

The two alternative improved algorithms make exchange between time and space. And they both do not change the time and space complexity of the original algorithm.

7 Conclusions and Future Works

We propose an improved algorithm to find the anti-block vital edge of a shortest path. And the constraint in the previous algorithm [8] is eliminated. Our algorithm runs in $O(m + n\log n)$ time which is faster than the one in [8] whose complexity is $O(mn)$.

In addition, Nardelli, Proietti and Widmayer [6] improved their algorithm using a linear time algorithm for the shortest path tree by means of a *transmuter* [9]. They gave an $O(m\alpha(m,n))$ time algorithm. Thus, Our algorithm can also be implemented in this way.

Bhosle and Gonzalez [2] found the replacement paths for all tree edges of a shortest path tree in $O(m + n\log n)$ time. Thus, our algorithm can be extended based on their technique.

References

[1] Bhosle, A.M., "Improved algorithms for replacement paths," *Operatjions Research Letters*, V33, pp. 459-466, 2005

[2] Bhosle, A.M., Gonzalez, T.F., "Algorithms for simple link failure recovery and related problems," *Journal of Graph Algorithms and Applications*, V8, N3, pp. 275-294, 2004

[3] Corley, H.W., Sha, D.Y., "Most vital links and nodes in weighted networks," *Operation Research Letters*, V1, pp. 157-161, 1982

[4] Fredman, M.L., Tarjan, R.E., "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM*, V34, N3, pp. 596-615, 1987

[5] Malik, K., Mittal, A.K., Gupta, S.K., "The k most vital arcs in the shortest path problem," *Operation Research Letters* V8, pp. 223-227, 1989

[6] Nardelli, E., Proietti, G., Widmayer, P., "A faster computation of the most vital edge of a shortest path between two nodes," *Information Processing Letters*, V79, N2, pp. 81-85, 2001

[7] Nardelli, E., Proietti, G., Widmayer, P., "Finding the detour critical edge of a shortest path between two nodes," *Information Processing Letters*, V67, N1, pp. 51-54, 1998

[8] Su, B., Xu, Q., Xiao, P., "Finding the anti-block vital edge of a shortest path between two nodes," In *Proceeding of COCOA 2007*, *Lecture Notes in Computer Science*, V4616, pp. 11-19, 2007

[9] Tarjan, R.E., "Applications of path compression on balanced tree," *Journal of the ACM*, V26, pp. 690-715, 1979