THE LONGEST DETOUR PROBLEM ON A SHORTEST PATH TREE*

Fu-Long Yeh, Shyue-Ming Tang, Yue-Li Wang and Ting-Yem Ho

Department of Information Management,

National Taiwan University of Science and Technology, Taipei City, Taiwan, R. O. C.

Email: ylwang@cs.ntust.edu.tw

ABSTRACT

In a biconnected graph, a *detour* from a vertex *u* to some destination vertex *s* is defined as an alternative shortest path from *u* to *s* when the edge (*u, v*) is not available in a shortest path <*u, v, ..., s*>. The *longest detour* (*LD*) *problem* is to find an edge (*u, v*), called the *detour-critical edge*, along a shortest path <*r,...,u,v,...,s*>, such that the removal of (*u, v*) may cause maximum increment of distance from *u* to *s*. The LD problem can be solved in O(*m* + *n log n*) time, where *m* and *n* denote the number of vertices and edges, respectively, in a graph. In this paper, we are concerned with the LD problem with respect to a shortest path tree of a graph. An O(*m* α(*m*, *n*)) time algorithm for finding a detour-critical edge in a shortest path tree is presented in this paper, where α is a functional inverse of Ackermann's function.

Keywords : Longest detour, Detour-critical edge, Shortest path, Biconnected graphs.

1. INTRODUCTION

Let *G*(*V, E*) be an undirected graph, where *V* and *E* are vertex set and edge set, respectively. A *weighted graph* is a graph in which every edge $e \in E$ is associated with a nonnegative real weight *w*(*e*) which can also be denoted as *w*(*x, y*) if *x* and *y* are two end vertices of *e* . The *length* of a path is the weight summation of edges in the path. A *shortest path* between vertices *r* and *s* in *G*, denoted as $P_G(r, s)$, is defined as a path with the shortest length from *r* to *s*. The *distance* between vertices *r* and *s*, denoted as $d_G(r, s)$, is the length of $P_G(r, s)$.

Let $P_G(r, s)$=<*r,...,u,v,...,s*> be the shortest path from *r* to *s*. A *detour* at vertex *u*, denoted as $P_{G-e}(u, s)$, is defined as a shortest path from *u* to *s* without using the edge *e=(u, v)*. The graph *G-e* is obtained by removing edge *e* from *G*. Notice that a detour is a shortest path from *u* to *s*, not *r* to *s*, in *G-e*. The *longest detour* (*LD*) *problem* is to find an edge *e*=(*u, v*), called the *detour-critical edge*, in $P_G(r, s)$ such that $d_{G-e}(u, s)$ minus $d_G(u, s)$ is maximum.

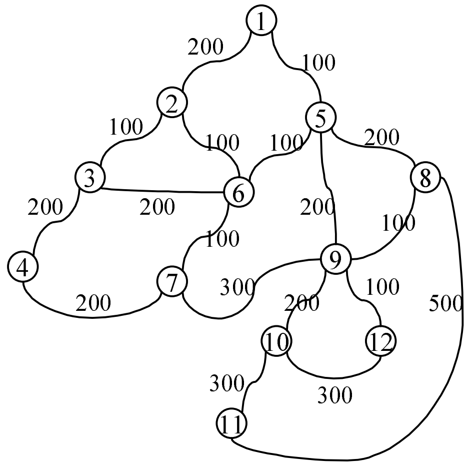

Figure 1. A weighted graph G.

For example, see Figure 1. $P_G(6,1)$ = <6, 5, 1> is the shortest path from 6 to 1. Paths <6,2,1> and <5,6,2,1> are the detours at vertices 6 and 5, respectively, with respect to $P_G(6,1)$. In other words, paths <6,2,1> and <5,6,2,1> are taken as the shortest paths toward vertex 1 when edges

* This work was supported by the National Science Council, Republic of China, under Contract NSC-89-2218-E011-019.

(6,5) and (5,1), respectively, are not available. The detour-critical edge of $P_G(6,1)$ is (5,1), because $d_{G-(5,1)}(5,1) - d_G(5,1) = 400 - 100 = 300$ is greater than $d_{G-(6,5)}(6,1) - d_G(6,1) = 300 - 200 = 100$.

The algorithm for finding detours, as well as determining the detour-critical edge, is important from the viewpoint of network management. Due to a sudden link failure from some node, a message must be transmitted through a detour from the node instead of the failed link. In [9], Nardelli et al. gave an $O(m + n \log n)$ algorithm for finding a detour-critical edge on a shortest path, where *m* and *n* are the number of vertices and edges, respectively, of a graph.

Let $S_G(s)$ be a tree of the shortest paths from vertex *s* to all other vertices in a graph *G*. The LD problem with respect to $S_G(s)$, called the *Tree Longest Detour* (*TLD*) *problem*, is to find a detour-critical edge (u,v) in $S_G(s)$, with *v* closer to *s* than *u*, such that the removal of (u,v) results in maximum increment of distance from *u* to *s*. By applying Nardelli's algorithm for every path in $S_G(s)$, this problem can be solved in $O(mn + n^2 \log n)$ time. We shall design an $O(m\ \alpha(m,n))$ time algorithm to conquer this problem, where α is a functional inverse of Ackermann's function.

In the past, the *shortest path* related problems have been studied widely [3,6,8,10]. The *most vital edge* (*MVE*) *problem*, or the *1-MVE problem*, which is a variation of the shortest path problem, has also been studied widely [2,7,5]. There are two definitions concerning the MVE problem. In [5], a *most vital edge* with respect to the minimum spanning tree of a graph is the edge that, when removed, results in the greatest weight increment of the minimum spanning tree. In [2] and [7], a *most vital edge* with respect to a shortest path is the edge whose removal results in the greatest increase in the distance between two end vertices. By comparing their definitions, we know that the detour-critical edge is different from the most vital edge with respect to a shortest path. The most vital edge of $P_G(r, s)$ is the same as the most vital edge of $P_G(s, r)$. However, the detour-critical edge of $P_G(r, s)$ is not necessarily the same as the detour-critical edge of $P_G(s, r)$.

The remainder of this paper is organized as follows. In Section 2, we introduce some notation used in this paper. In Section 3, we propose an $O(m\ \alpha(m,n))$ time algorithm for solving the TLD problem. Section 4 contains the concluding remarks.

2. NOTATION

Before describing our algorithm, we define some notation which will be used throughout this paper. Most of them are also used in [9].

Let $S_G(s)$ be a shortest path tree of *G* rooted at vertex *s*. An edge in $S_G(s)$ is called a *tree edge*, and a *nontree edge* if it is not a tree edge in $S_G(s)$. Clearly, any detour from a vertex in the graph must make use of some nontree edge in order to arrive at *s* again. For example, Figure 2 is $S_G(1)$ of the graph in Figure 1. Nontree edge (6, 2) is included in the detour from vertex 6 when tree edge (6, 5) is not available.

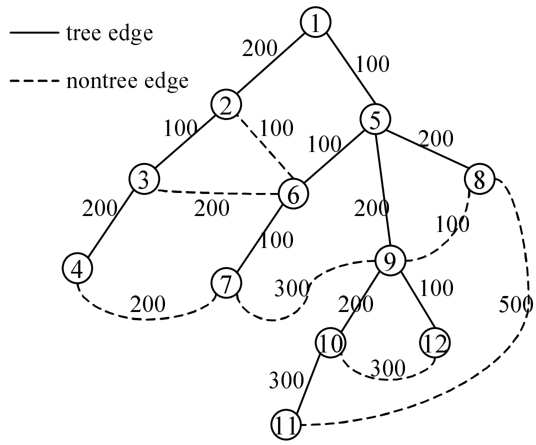

Figure 2. $S_G(1)$ of graph G.

It follows throughout this paper that an edge (u, v) in $S_G(s)$ means that the latter vertex *v* is closer to *s* than *u*. The former vertex *u* is called the *detour starting vertex*. Based on the property of a shortest path tree, we have the following lemma.

Lemma 1 *Let e=(u, v) be an edge in $S_G(s)$ and u be a detour starting vertex. There exists a detour from u to the destination vertex s which contains exactly one nontree edge with respect to $S_G(s)$.*

Proof : Let *(x, y)* be the first nontree edge encountered in the detour from *u* to *s*. Clearly, the path from *y* to *s* in $S_G(s)$ is a shortest path between *y* and *s*. If there exists another nontree edge (x', y') in the detour, then the path from *y* to *s* which pass through (x', y') must be longer than or equal to the path from *y* to *s* in $S_G(s)$. Thus, the lemma follows. Q. E. D.

According to Lemma 1, we call the only nontree edge in a detour the *crossing edge* of the detour. Note that the detour from a vertex may be more than one. In case of multiple detours, their length must be equal. The length of a detour from u to s is formulated as :

$$d_{G-e}(u, s) = d_{G-e}(u, x) + w(x, y) + d_{G-e}(y, s),$$

where $e=(u, v)$ is a tree edge in $S_G(s)$ and (x, y) is the crossing edge of vertex u. In fact, $d_{G-e}(u, x) = d_G(u, x)$, since x belongs to the subtree of $S_G(s)$ rooted at u. Moreover, $d_{G-e}(y, s) = d_G(y, s)$, since y belongs to the set of vertices reachable from s without passing through edge e in $S_G(s)$. Thus, we have

$$d_{G-e}(u,s) = d_G(u,x) + w(x,y) + d_G(y,s) .$$

Let z be the lowest common ancestor of the two vertices of a nontree edge (x, y). A *fundamental cycle*, denoted as $C(x, y)$, is the cycle which consists exactly of $P_G(z, x)$, (x, y) and $P_G(y, z)$. The *K-value* of $C(x, y)$, denoted as $K(x, y)$, is the total length of a close walk from s to x, then to y, finally from y to s. That is,

$$K(x, y) = d_G(s, x) + w(x, y) + d_G(y, s).$$

For example, see Figure 2 again. $K(2, 6) = d_G(1, 2) + w(2, 6) + d_G(6, 1) = 500$, and $K(7, 9) = d_G(1, 7) + w(7, 9) + d_G(9, 1) = 900$. Notice that the lowest common ancestor of the two vertices of a nontree edge (x, y) may be the root of $S_G(s)$. In case of identity, $K(x, y)$ is the total length of fundamental cycle $C(x, y)$.

Although a fundamental cycle is one-to-one corresponding to a nontree edge, multiple fundamental cycles may cover the same tree edges of $S_G(s)$. For example, tree edge (6, 5) in Figure 2 is covered by four fundamental cycles, i.e., $C(2, 6)$, $C(3, 6)$, $C(4, 7)$ and $C(7, 9)$. It is obviously true that if G is biconnected, then each tree edge of $S_G(s)$ must be covered by at least one fundamental cycle. Let (u, v) be the unavailable edge in $S_G(s)$, and (x, y) be a crossing edge of vertex u. Recall that a crossing edge is the only nontree edge in a detour. The fundamental cycle $C(x, y)$ which covers both (x, y) and (u, v) is called a *detour cycle* of vertex u. In Figure 2, C(2, 6) is a detour cycle of vertex 6 in $S_G(1)$.

Lemma 2 *Let F_e be the set of fundamental cycles which cover edge e in $S_G(s)$. $C(x, y)$ is a detour cycle of e if and only if the K-value of $C(x, y)$ is minimum among all fundamental cycles in F_e .*

Proof : Let (x, y) be the crossing edge of a fundamental cycle $C(x, y) \in F_e$,where $e=(u, v)$, and assume without loss of generality that u is in the shortest path from s to x. Since $d_G(s, x) = d_G(s, u) + d_G(u, x)$, the K-value of $C(x, y)$ can be derived as follows :

$$K(x, y) = d_G(s, x) + w(x, y) + d_G(y, s)$$
$$= d_G(s, u) + d_G(u, x) + w(x, y) + d_G(y, s).$$

In the above formula, $d_G(s, u)$ is the common item of the K-values of all fundamental cycles in F_e. If the K-value of $C(x, y)$ is minimum among all fundamental cycles in F_e , then $d_G(u, x) + w(x, y) + d_G(y, s)$ must be minimum and (x, y) is the crossing edge of a detour from u.

Conversely, suppose that (x, y) is a crossing edge of vertex u. The value of $d_G(u, x) + w(x, y) + d_G(y, s)$ is the length of a detour from u to s. It must be minimum among all feasible paths from u to s in $G - e$. Therefore, the K-value of $C(x,y)$ is also minimum among all fundamental cycles in F_e . Q. E. D.

We use an example to illustrate Lemma 2. In Figure 2, fundamental cycles which cover edge (6,5) in $S_G(1)$ (i.e., $F_{(6,5)}$) are $C(2,6)$, $C(3,6)$, $C(4,7)$ and $C(7,9)$. $C(2,6)$ is a detour cycle of vertex 6 because $K(2,6)$ is the smallest among $K(2,6)$, $K(3,6)$, $K(4,7)$ and $K(7,9)$.

Lemma 2 shows that we can compute the detour length from the K-value of a detour cycle. That is, the detour length from a vertex u equals $d_G(u, x) + w(x, y) + d_G(y, s) = K(x, y) - d_G(s, u)$, where (x, y) is a crossing edge of vertex u. Let $I(u)$ be the *distance increment* from a detour starting vertex u to the destination vertex s. To find a detour-critical edge $e=(u,v)$ of $S_G(s)$ is to find the maximum $I(u)$ among all detour starting vertices u in $S_G(s)$. Lemma 3 provides a formula to compute $I(u)$ efficiently.

Lemma 3 *Let u be a detour starting vertex in $S_G(s)$. Then, $I(u) = K(x,y) - 2\, d_G(s,u)$, where (x,y) is the crossing edge of a detour from vertex u.*

Proof : Let $e = (u,v)$ be a tree edge in $S_G(s)$.

$$I(u) = d_{G-e}(u, s) - d_G(u, s)$$
$$= d_G(u, x) + w(x, y) + d_G(y, s) - d_G(s, u)$$
$$= K(x, y) - d_G(s, u) - d_G(s, u)$$
$$= K(x, y) - 2\, d_G(s, u).$$

Q. E. D.

3. AN EFFICIENT ALGORITHM FOR SOLVING THE TLD PROBLEM

To determine a detour-critical edge of $S_G(s)$, we should compute the distance increment $I(u)$ for every vertex u in $S_G(s)$ except vertex s. Intuitively, since there are at most O(m) fundamental cycles which may cover a detour starting vertex with respect to $S_G(s)$, the computation of $I(u)$ needs O(m) time for vertex u. It requires O(mn) time to compute $I(u)$ for all the vertices in $S_G(s)$. Therefore, it takes O(mn) time to determine a detour-critical edge of $S_G(s)$ by using the naive algorithm. However, in the following, we propose an O(m $\alpha(m, n)$) algorithm to solve this problem.

To design an efficient algorithm for solving the TLD problem, we need a data structure, called a *transmuter*, which was introduced in [10]. A transmuter is a directed acyclic graph that represents the relation between the detour starting vertices and the fundamental cycles in $S_G(s)$. In a transmuter, one source (node of in-degree zero) represents a detour starting vertex in $S_G(s)$, one sink (node of out-degree zero) represents a fundamental cycle in $S_G(s)$, and every intermediate node has at least two out-degrees as well as at least two in-degrees. We label each source node with its corresponding vertex and each sink node with its corresponding nontree edge of $S_G(s)$. The fundamental properties of a transmuter are as follows. (i) There is a directed edge from a source node u to a sink node $C(x, y)$ in the transmuter if and only if vertex u is covered by fundamental cycle $C(x, y)$ in G. (ii) When two or more detour starting vertices share two or more fundamental cycles, there exists a common intermediate node in the paths from the corresponding source nodes to the corresponding sink nodes. For example, the transmuter corresponding to $S_G(1)$ is shown as follows.

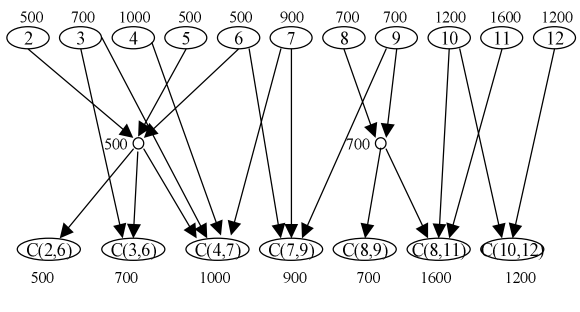

Figure 3. Transmuter for $S_G(1)$.

Since vertex 3 is covered by C(3,6), there is a directed edge from the source node 3 to the sink node C(3,6). All of the vertices 2, 5 and 6 are covered by all of the fundamental cycles C(2,6), C(3,6) and C(4,7). Therefore, there exists a common intermediate node in the paths from the source nodes to the sink nodes.

In [11], Tarjan has described an O(m $\alpha(m,n)$) time algorithm to construct a transmuter for a shortest path tree with respect to a graph of size m and order n. Meanwhile, it has been proved that a transmuter has O(m $\alpha(m,n)$) nodes and edges. Given a transmuter, we can efficiently determine a detour cycle, as well as a crossing edge, for each vertex in $S_G(s)$.

We can compute $I(u)$ for every detour starting vertex u in $S_G(s)$ by using the transmuter of $S_G(s)$. Let each node x of the transmuter have an associated value $A(x)$. If x is a sink node, then $A(x)$ is equal to the K-value of the fundamental cycle x; otherwise, $A(x)$ = $Min_{y\in N(x)}\{A(y)\}$, where $N(x)$ is the immediate successors of x. In Figure 3, The value beside each node is the associated value of that node. For a source node u, $A(u)$ is the K-value of a detour cycle of u. Thus, $I(u)$ = $A(u)$ - 2 $d_G(u,s)$ for every detour starting vertex u in $S_G(s)$.

Now, we are in a position to describe our algorithm for solving the TLD problem.

Algorithm Find_DCE

Input: A shortest path tree rooted at vertex s, $S_G(s)$, of a biconnected graph $G(V,E)$.

Output: A detour-critical edge of $S_G(s)$.

Method:

Step 1. For every nontree edge $(x, y)\in S_G(s)$, compute the K-value of $C(x, y)$, where
$K(x, y) = d_G(s, x) + w(x, y) + d_G(y, s)$.

Step 2. Construct a transmuter of $S_G(s)$.

Step 3. Obtain the associated value of $A(x)$ for each node x in the transmuter.

Step 4. For each source node u in the transmuter, ompute $I(u) = A(u) - 2\, d_G(u, s)$.

Step 5. Find a detour-critical edge (u, v) of $S_G(s)$, where $I(u)$ is maximum among all the associated values of the source nodes in the transmuter.

End of Algorithm Find_DCE

We also use Figure 1 as an example to illustrate **Algorithm Find_DCE**. The result of Steps 1 to 3 are shown as Figure 3. Step 4 computes $I(u)$ for each vertex u. We can see that $I(2) = A(2) - 2\ d_G(2, 1) = 500 - 400 = 100$. The values of $I(3), I(4), \ldots, I(12)$ are 100, 0, 300, 100, 300, 100, 100, 200, 0 and 400. Therefore, the detour-critical edge of $S_G(1)$ is edge $(12,9)$ since $I(12) = 400$ makes the largest distance increment when edge $(12,9)$ is unavailable.

Step 1 takes O(m) time to compute $K(x,y)$ for all of the nontree edges in $S_G(s)$. Step 2 requires O($m\ \alpha(m,n)$) time to construct the transmuter of $S_G(s)$ by using Tarjan's algorithm. Step 3 also takes O($m\ \alpha(m,n)$) time to obtain the associated values of all the nodes in the transmuter. Obviously, both Steps 4 and 5 require O(n) time. Therefore, the time complexity of **Algorithm Find_DCE** is O($m\ \alpha(m,n)$).

By summarizing above description, we have the following theorem.

Theorem 4. *Algorithm Find_DCE can solve the TLD problem in* $O(m\ \alpha(m, n))$ *time.*

4. CONCLUDING REMARKS

The TLD problem has many interesting properties. For example, there may exist multiple detour-critical edges in a shortest path tree. There may also exist multiple detours or detour cycles from a detour starting vertex with respect to a shortest path tree. With minor modification, our algorithm can find all detour cycles of a detour starting vertex, as well as all detour-critical edges in a shortest path tree.

In general, we extend the LD problem from a single shortest path to a shortest path tree rooted at a vertex of a given graph. Assume the LD problem of one shortest path to be a "one-to-one" fashion, the TLD problem will be a "many-to-one" fashion. The former can be viewed as a special case of the latter. In the near future, we shall focus our study on an efficient algorithm for finding a detour-critical edge with respect to all paired shortest paths in an undirected graph. That problem is in a fashion of "many-to-many". A parallel algorithm dedicated to the TLD problem is another topic that needs our effort.

References

[1] Bras88 G. Brassard and P. Bratley, *Algorithmics: Theory and Practice,* Prentice-Hall Inc., 1988, pp. 30-34.

[2] H. W. Corley and D. Y. Sha, Most vital links and nodes in weighted networks, *Operations Research Letters,* Vol.1, No.4, 1982, pp. 157-160.

[3] E. W. Dijkstra, A note on two problems in connection with graphs, *Numeric Mathematics,* 1, 1959, pp. 269-271.

[4] M. L. Fredman and R. E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *Journal of the ACM,* Vol. 34, 1987, pp. 596-615.

[5] L.-H. Hsu, et. al., Finding the most vital edge with respect to minimum spanning tree in weighted graphs, *Information Processing Letters,* 39, 1991, pp. 277-281.

[6] D. E. Knuth, A generalization of Dijkstra algorithm, *Information Processing Letters,* 6, 1977, pp.1-5.

[7] K. Malik, A. K. Mittal, and S. K. Gupta, The k-most vital arcs in the shortest path problem, *Information Processing Letters,* 8, 1989, pp. 223-227.

[8] E. F. Moore, The shortest path through a maze, *Proc. International Symposium on Switching Theory,* Harvard Univ. Press, Cambridge, 1959, pp. 285-292.

[9] E. Nardelli, G. Proietti and P. Widmayer, Finding the detour-critical edge of a shortest path between two nodes, *Information Processing Letters,* 67, 1998, pp. 51-54.

[10] R. E. Tarjan, Sensitivity analysis of minimum spanning trees and shortest path trees , *Information Processing Letters,* 14, 1982, pp. 30-33.

[11] R. E. Tarjan, Applications of path compression on balanced trees, *Journal of the ACM,* Vol. 26, No. 4, 1979, pp. 690-715.