Discrete Optimization

The tree longest detour problem in a biconnected graph

Fu-Long Yeh a, Shyue-Ming Tang a, Yue-Li Wang a,*, Ting-Yem Ho b

a Department of Information Management, National Taiwan University of Science and Technology, 43, Section 4, Kee-Lung Road, Taipei 106, Taiwan, ROC
b Department of Public Finance and Tax Administration, National Taipei College of Business, Taipei 100, Taiwan, ROC

Received 18 April 2001; accepted 4 March 2002

Abstract

In a biconnected graph, a detour is the best alternative path from a detour-starting vertex to the destination vertex. A detour-starting vertex is the vertex from which the original shortest path is changed. The longest detour (LD) problem is to find a detour-critical edge in a shortest path such that the removal of the edge results in the maximum distance increment. In this paper, we deal with the LD problem with respect to a shortest path tree. An efficient algorithm which takes $O(m \alpha(m,n))$ time for finding a detour-critical edge in a shortest path tree is proposed, where $\alpha$ is a functional inverse of Ackermann’s function.

© 2003 Elsevier B.V. All rights reserved.

Keywords: Graph theory; Tree longest detour problem; Detour-critical edge; Shortest path tree; Biconnected graphs

1. Introduction

Let $G(V,E)$ be a weighted graph, where $V$ and $E$ are vertex set and edge set, respectively. Every edge $e = (u,v)$ in $E$ is associated with a nonnegative real weight $w(e)$ or $w(u,v)$. The length of a path is the sum of the weights of the edges in the path. A shortest path between vertices $r$ and $s$ in $G$, denoted by $P_G(r,s)$, is defined as a path with the shortest length from $r$ to $s$. The distance between vertices $r$ and $s$, denoted by $d_G(r,s)$, is the length of the shortest path $P_G(r,s)$.

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

Corresponding author.
E-mail address: ylwang@cs.ntust.edu.tw (Y.-L. Wang).

Let $P_G(v_0,v_p) = \langle v_0, v_1, v_2, \ldots , v_p \rangle$ be a shortest path from $v_0$ to $v_p$ in a biconnected graph $G$. A detour from $v_i$ to $v_p$, denoted by $P_{G-e}(v_i,v_p)$, is a shortest path from $v_i$ to $v_p$ that does not contain the edge $e = \langle v_i, v_{i+1} \rangle$, where $i \neq p$. The graph $G - e$ is obtained by removing edge $e$ from $G$. Notice that a detour is a shortest path from $v_i$ to $v_p$, not $v_0$ to $v_p$, in $G - e$. The longest detour (LD) problem is to find an edge $\langle v_i, v_{i+1} \rangle$, called the detour-critical edge, along the path such that the removal of the edge results in a maximum increase of the distance from $v_i$ to $v_p$. For example, see Fig. 1. $P_G(7,1) = \langle 7,6,5,1 \rangle$ is the shortest path from 7 to 1. Path $\langle 7,9,5,1 \rangle$ is the detour from vertex 7 to vertex 1 if edge $\langle 7,6 \rangle$ fails. Path $\langle 6,2,1 \rangle$ is the detour from vertex 6 to vertex 1 if edge $\langle 6,5 \rangle$ fails. Path $\langle 5,6,2,1 \rangle$ is the detour from vertex 5 to vertex 1 if edge $\langle 5,1 \rangle$ fails. The detour-critical edge
The shortest paths from vertex \( s \) to all other vertices in a graph \( G \) form a shortest path tree, denoted by \( T \). In this paper, we extend the LD problem from a shortest path to a shortest path tree. The LD problem with respect to a shortest path tree \( T \), called the tree longest detour (TLD) problem, is to find a detour-critical edge in \( T \). That is, to find an edge \( e = (u, v) \), with \( v \) closer to \( s \) than \( u \), such that \( d_{G-e}(u, s) - d_G(u, s) \) is maximum.

In [8], Nardelli et al. gave an \( O(m + n \log n) \) time algorithm for finding a detour-critical edge in a shortest path, where \( m \) and \( n \) are the number of edges and vertices, respectively, in a graph. By naively applying Nardelli’s algorithm for \( n - 1 \) shortest paths in \( T \), the TLD problem can be solved in \( O(mn + n^2 \log n) \) time. We propose an algorithm to evaluate every edge in \( T \) more efficiently, and only take \( O(mx(m, n)) \) time to find the detour-critical edge in \( T \), where \( x \) is a functional inverse of Ackermann’s function [2].

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

2. Basic notation and main idea

Before describing our algorithm, we define some notation that will be used throughout this paper. Most of them are also used in [8].

Let \( T \) be a shortest path tree rooted at vertex \( s \) in \( G \). An edge of \( G \) that is in \( T \) is called a tree edge, while an edge of \( G \) that is not in \( T \) is called a non-tree edge. A shortest path tree rooted at vertex 1 of the example graph is shown in Fig. 2.

For a tree edge \((u, v)\) in \( T \), we call vertex \( u \) the detour-starting vertex if vertex \( v \) is the parent vertex of vertex \( u \). See Fig. 3 for an illustration. Since \( T \) is disconnected after any tree edge is removed, a detour from a detour-starting vertex in \( T \) must contain some non-tree edge in order to arrive at \( s \) again. Let \( S \) be the subtree rooted at \( u \) in \( T \) and let \( T - S \) be a subgraph obtained by removing \( S \) from \( T \). See Fig. 3 for an illustration. Based on the
Proof. Let $D$ be a detour from $u$ to $s$. Since $T$ is disconnected after tree edge $(u,v)$ is removed, $D$ must contain at least one nontree edge $(x,y)$ such that vertices $x$ and $y$ are in the vertex sets of $S$ and $T - S$, respectively. In fact, $d_T(u,x) = d_c(u,x)$ since $x$ belongs to the subtree rooted at $u$ in $T$. Meanwhile, $d_T(y,s) = d_c(y,s)$ since $y$ belongs to the set of vertices reachable from $s$ without passing through edge $(u,v)$ in $T$. Thus, the value of $d_T(u,x) + w(x,y) + d_T(y,s)$ is also minimum and must be equal to the length of $D$. That is, the path that contains only one nontree edge $(x,y)$ is also a detour from vertex $u$. Therefore, the lemma follows. □

According to Lemma 1, we call the unique nontree edge in a detour the crossing edge. For example, when tree edge $(6,5)$ in Fig. 2 is not available, the crossing edge of the detour from vertex 6 is the nontree edge $(6,2)$.

Notice that there may exist multiple detours from a detour-starting vertex, while all of them have the same length. The length of a detour from a detour-starting vertex $u$ to root vertex $s$ is formulated as

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

where $e = (u,v)$ is a faulty edge in $T$ and $(x,y)$ is the crossing edge of the detour from vertex $u$.

It turns out that a crossing edge is a nontree edge that can minimize the detour length. Furthermore, only nontree edges that bridge $S$ and $T - S$ have to be taken into consideration in determining the crossing edge.

Any nontree edge can create exactly one cycle when it is added to $T$. Such a cycle is called a fundamental cycle of $G$ with respect to $T$ [9]. Since every spanning tree of graph $G$ contains $n - 1$ edges, there are $m - n + 1$ fundamental cycles with respect to $T$. Let $C(x,y)$ denote the fundamental cycle that is created by adding nontree edge $(x,y)$ to $T$.

A nontree edge can determine a unique fundamental cycle, while the nontree edge may also be the crossing edge of some detour. It is a natural thing to associate a fundamental cycle with a detour. Let $u$ be a detour-starting vertex in $T$ and let $(x,y)$ be the crossing edge of a detour from vertex $u$. Then, the fundamental cycle $C(x,y)$ is called a detour cycle of vertex $u$. For example, tree edge $(6,5)$ in Fig. 2 is covered by four fundamental cycles $C(2,6)$, $C(3,6)$, $C(4,7)$, and $C(7,9)$. Since nontree edge $(2,6)$ is the crossing edge of a detour from vertex 6, $C(2,6)$ is the detour cycle of vertex 6.

Since there are $n - 1$ detour-starting vertices and $m - n + 1$ fundamental cycles with respect to $T$, a number of vertices may have the same detour cycle, while a nontree edge of some fundamental cycle may not be the crossing edge of any vertex.
In order to determining the detour cycle efficiently for every detour-starting vertex, we have to set a value to each fundamental cycle.

The \( K \)-value of a fundamental cycle \( C(x, y) \), denoted by \( K(x, y) \), is the total length of a close walk from the root \( s \) to \( x \), then to \( y \), and finally from \( y \) to \( s \). That is, \( K(x, y) = d_T(s, x) + w(x, y) + d_T(y, s) \).

For example, see Fig. 2 again. \( K(6, 2) = d_T(1, 6) + w(6, 2) + d_T(2, 1) = 200 + 100 + 200 = 500 \), and \( K(9, 7) = d_T(1, 9) + w(9, 7) + d_T(7, 1) = 200 + 300 + 300 = 800 \).

For a detour-starting vertex \( u \), a number of fundamental cycles may cover it. Lemma 2 proves that the \( K \)-value of a detour cycle is the smallest one.

**Lemma 2.** Let \((u, v)\) be a faulty edge and \( u \) be a detour-starting vertex in \( T \). \( C(x, y) \) is a detour cycle of the detour from vertex \( u \) if and only if \( C(x, y) \) has the minimum \( K \)-value among all fundamental cycles that cover \((u, v)\).

**Proof.** Let \((x, y)\) be the nontree edge that creates a fundamental cycle \( C(x, y) \) with \( x \) and \( u \) in the same subtree of \( T \). Since \( d_T(s, x) = d_T(s, u) + d_T(u, x) \), the \( K \)-value of \( C(x, y) \) can be derived as follows:

\[
K(x, y) = d_T(s, x) + w(x, y) + d_T(y, s) = d_T(s, u) + d_T(u, x) + w(x, y) + d_T(y, s).
\]

In this formula, \( d_T(s, u) \) is the common item of the \( K \)-values for all fundamental cycles that cover \((u, v)\). If \( C(x, y) \) has the minimum \( K \)-value among all fundamental cycles, then the detour length \( d_T(u, x) + w(x, y) + d_T(y, s) \) is also minimum. That is, edge \((x, y)\) is the crossing edge and \( C(x, y) \) is the detour cycle of a detour from vertex \( u \).

Conversely, suppose that \((x, y)\) is the crossing edge and \( C(x, y) \) is the detour cycle of a detour starting from vertex \( u \). By definition, the detour has the shortest path from vertex \( u \) to vertex \( s \) in graph \( G - e \). That is, \( d_T(u, x) + w(x, y) + d_T(y, s) \) is minimum. From the formula \( K(x, y) = d_T(s, u) + d_T(u, x) + w(x, y) + d_T(y, s) \), the detour cycle \( C(x, y) \) also have the minimum \( K \)-value among all fundamental cycles that cover \((u, v)\).

We use an example to illustrate Lemma 2. In Fig. 2, there are four fundamental cycles that cover edge \((6, 5)\), i.e., \( C(6, 2), C(6, 3), C(7, 4), \) and \( C(7, 9) \). \( C(6, 2) \) is the detour cycle of a detour from vertex \( 6 \) since \( K(6, 2) = 500 \) is less than \( K(6, 3) = 700, K(7, 4) = 1000 \) and \( K(7, 9) = 800 \).

By definition, the length of the detour from vertex \( u \) is \( d_T(u, x) + w(x, y) + d_T(y, s) = K(x, y) - d_T(s, u) \), where \((x, y)\) is the crossing edge of the detour. The distance increment of a detour-starting vertex \( u \), denoted by \( \text{Inc}(u) \), is the increase of distance from vertex \( u \) to root \( s \). That is, \( \text{Inc}(u) = d_{G-e}(u, s) - d_T(u, s) \). The TLD problem is to find the detour-critical edge \( e = (u, v) \) on \( T \), such that \( d_{G-e}(u, s) - d_T(u, s) \), i.e., \( \text{Inc}(u) \), is maximum. Lemma 3 provides a formula to compute \( \text{Inc}(u) \) efficiently.

**Lemma 3.** Let \( u \) be a detour-starting vertex on \( T \). Then, \( \text{Inc}(u) = K(x, y) - 2d_T(s, u) \), where \((x, y)\) is the crossing edge of a detour from vertex \( u \).

**Proof.** Let \( e = (u, v) \) be a tree edge in \( T \), with \( v \) closer to \( s \) than \( u \).

\[
\text{Inc}(u) = d_{G-e}(u, s) - d_T(u, s) = d_T(u, x) + w(x, y) + d_T(y, s) - d_T(s, u) = K(x, y) - d_T(s, u) - d_T(s, u) = K(x, y) - 2d_T(s, u).
\]

To determine a detour-critical edge of \( T \), we have to compute \( \text{Inc}(u) \) for every detour-starting vertex \( u \) in \( T \). Since there are at most \( m - n + 1 \) fundamental cycles which may cover a detour-starting vertex with respect to \( T \), it seems that, intuitively, determining the detour cycle for every detour-starting vertex takes \( O(m) \) time. As a result, the total time-complexity is \( O(mn) \) for finding a detour-critical edge in \( T \). In the following section, we shall propose a more efficient algorithm to solve the TLD problem.

### 3. An efficient algorithm for solving the TLD problem

Intuitively, it takes \( O(m) \) time to determine the detour cycle of a detour-starting vertex. However,
many computation works are redundant for every detour-starting vertex. To avoid such repeated computation, we use a special data structure, called transmuter, to associate a detour-starting vertex with its detour cycle.

A transmuter, which was introduced in [10], is a directed acyclic graph that represents the relationships between detour-starting vertices and fundamental cycles with respect to \( T \). There are three types of nodes in a transmuter, i.e., source node, sink node, and intermediate node. A source node (i.e., a node with no in-coming edge) is labeled by a detour-starting vertex, while a sink node (i.e., a node with no out-going edge) is labeled by a fundamental cycle. A detour-starting vertex \( u \) is related to a fundamental cycle \( C(x, y) \) if tree edge \((u, v)\) is covered by the fundamental cycle \( C(x, y) \), where \( v \) is the parent vertex of \( u \) in \( T \). If two or more detour-starting vertices are related to the same set of fundamental cycles, then there is an intermediate node with them. The directed edges of a transmuter are constructed by the following rules:

(i) There is a directed edge from a source node \( u \) to a sink node \( C(x, y) \) if \( u \) is related to \( C(x, y) \) and \( u \) has no corresponding intermediate node with \( C(x, y) \).

(ii) For a source node \( u \), if it has a corresponding intermediate node \( p \), then there is a directed edge from \( u \) to \( p \).

(iii) For a sink node \( C(x, y) \), if it has a corresponding intermediate node \( p \), then there is a directed edge from \( p \) to \( u \).

Notice that an intermediate node has at least two in-coming edges and at least two out-going edges. For example, the transmuter corresponding to the example shortest path tree is shown in Fig. 4. Since vertex 6 is related to fundamental cycle \( C(7, 9) \) and no other vertex is related to a common set (with vertex 6) of fundamental cycles that contains \( C(7, 9) \), there is a directed edge from the source node 6 to the sink node \( C(7, 9) \) in the transmuter. The detour-starting vertices 2, 5, and 6 are related to the same set of the fundamental cycles \( C(2, 6), C(3, 6), \) and \( C(4, 7) \). Thus, there exists an intermediate node in the paths from the source nodes 2, 5, and 6 to the sink nodes \( C(2, 6), C(3, 6), \) and \( C(4, 7) \).

In [11], Tarjan has described an \( O(mz(m, n)) \) time algorithm to construct a transmuter for a shortest path tree with respect to a graph of \( m \) edges and \( n \) vertices, where \( z(m, n) \) is a functional inverse of Ackermann’s function. Note that \( z(m, n) \) is an extremely slow growing function (for \( m, n < 2^{65536}, z(m, n) < 5 \)). Meanwhile, it has been proved that a transmuter has \( O(mz(m, n)) \) nodes and edges. Given a transmuter, we can efficiently determine the crossing edge, as well as the \( K \)-value of a detour cycle, for every detour-starting vertex in \( T \).

We can compute \( \text{Inc}(u) \) for every detour-starting vertex \( u \) on \( T \) by using the transmuter constructed from \( T \). Let each node \( x \) of the transmuter have an associated value \( A(x) \). Initially, for a sink node \( x, x = C(x, y) \), the value of \( A(x) \) is the \( K \)-value of the fundamental cycle \( C(x, y) \), i.e., \( K(x, y) \). The value of \( K(x, y) \) can be computed by the formula

\[
K(x, y) = \text{Inc}(x) + \text{Inc}(y) - 2 \cdot \text{Inc}(p)
\]
The results of computation

Table 1

<table>
<thead>
<tr>
<th>Detour-starting vertex u</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crossing edge (x,y)</td>
<td>(2,6)</td>
<td>(3,6)</td>
<td>(4,7)</td>
<td>(6,2)</td>
<td>(6,2)</td>
<td>(7,9)</td>
<td>(8,9)</td>
<td>(9,8)</td>
<td>(10,12)</td>
<td>(11,8)</td>
<td>(12,10)</td>
</tr>
<tr>
<td>A(u)</td>
<td>500</td>
<td>700</td>
<td>1000</td>
<td>500</td>
<td>500</td>
<td>800</td>
<td>600</td>
<td>600</td>
<td>1000</td>
<td>1500</td>
<td>1000</td>
</tr>
<tr>
<td>dy(u,s)</td>
<td>200</td>
<td>300</td>
<td>500</td>
<td>100</td>
<td>200</td>
<td>300</td>
<td>300</td>
<td>200</td>
<td>400</td>
<td>700</td>
<td>300</td>
</tr>
<tr>
<td>Inc(u)</td>
<td>100</td>
<td>100</td>
<td>0</td>
<td>300</td>
<td>100</td>
<td>200</td>
<td>0</td>
<td>200</td>
<td>200</td>
<td>100</td>
<td>400</td>
</tr>
</tbody>
</table>
time to compute $K(x,y)$ for all of them. Step 2 requires $O(mz(m,n))$ time to construct the transmuter from $T$ by using the algorithm in [11]. Since a transmuter contains $O(mz(m,n))$ nodes and edges, Step 3 also takes $O(mz(m,n))$ time to compute the associated value $A(x)$ for every node $x$ in the transmuter. Both Steps 4 and 5 can be done in $O(n)$ time. Therefore, the time-complexity of Algorithm Find_DCE is $O(mz(m,n))$.

By summarizing above description, we have the following theorem.

**Theorem 1.** Algorithm Find_DCE can correctly solve the TLD problem in $O(mz(m,n))$ time.

### 4. Concluding remarks

We have presented an $O(mz(m,n))$ time algorithm to solve the TLD problem in a biconnected graph. A special data structure, called transmuter, is applied in the algorithm. By using the transmuter, we can determine the detour cycle of every detour-starting vertex more efficiently. Then, by comparing the distance increment of every detour-starting vertex, the detour-critical edge of the shortest path tree is found.

The TLD problem has many interesting properties. From the above example, we can figure out that there may exist multiple detour-critical edges on a shortest path tree, since different faulty edges may result in the same distance increment. There may also exist multiple detours from a detour-starting vertex with equal distance to the destination vertex. With a minor modification, our algorithm can find all detour-critical edges on a shortest path tree, as well as all detours from a detour-starting vertex.

The underlying graph is assumed to be biconnected, so that for each faulty edge on the shortest path tree, at least one alternative path leading to root exists. In case of faulty vertex, the TLD problem has to be redefined. Intuitively, our algorithm still works for the TLD problem with faulty vertex.

We extend the LD problem from a single shortest path to a shortest path tree rooted at a vertex of a biconnected graph. The LD problem on a shortest path is a “one-to-one” type problem, while the TLD problem is a “many-to-one” type problem. The former can be viewed as a special case of the latter. For a more generalization, the problem to find a detour-critical edge with respect to all-pairs shortest paths can be viewed as “many-to-many” type problem. We are very interested in this problem.

### References