# Solving the Replacement Paths Problem for Planar Directed Graphs in O(n log n) Time

Christian Wulff-Nilsen

**Technical Report no. 09/03**
**ISSN: 0107-8283**

**Dept. of Computer Science**
University of Copenhagen • Universitetsparken 1
DK-2100 Copenhagen • Denmark

# Solving the Replacement Paths Problem for Planar Directed Graphs in $O(n \log n)$ Time

Christian Wulff-Nilsen [*]

June 14, 2009

**Abstract**

In a graph $G$ with non-negative edge lengths, let $P$ be a shortest path from a vertex $s$ to a vertex $t$. We consider the problem of computing, for each edge $e$ on $P$, the length of a shortest path in $G$ from $s$ to $t$ that avoids $e$. This is known as the *replacement paths problem*. We give a linear-space algorithm with $O(n \log n)$ running time for $n$-vertex planar directed graphs. The previous best time bound was $O(n \log^2 n)$.

## 1 Introduction

Computing shortest paths in graphs is a classical problem in combinatorial optimization with applications in numerous areas such as communication networks. These networks are in general not static but may change due to link failures. In such cases, alternative lines of communication need to be established and it may be of interest to determine the "quality" of such lines.

This motivates the *replacement paths problem (RPP)*: given two vertices $s$ and $t$ in a graph $G$ with non-negative edge lengths and given a shortest path $P$ (the line of communication) in $G$ from $s$ to $t$, compute, for each edge $e$ on $P$, the length of a shortest path in $G$ from $s$ to $t$ that avoids $e$ (if no such path exists, the length is defined to be infinite). More motivation for the RPP is given in [3].

[*]Department of Computer Science, University of Copenhagen, `koolooz@diku.dk`, `http://www.diku.dk/hjemmesider/ansatte/koolooz/`

The RPP is a well studied problem. For undirected graphs with $m$ edges and $n$ vertices, algorithms are known with running time $O(m + n \log n)$ [7] and $O(m\alpha(m, n))$ [8], respectively (the latter applying to a stronger model of computation).

The directed case is harder since it has an $\Omega(m\sqrt{n})$ lower bound [4]. The fastest known algorithm is the trivial one that removes each edge on shortest path $P$ in turn and applies Dijkstra's algorithm to the resulting graph. This gives a time bound of $O(mn + n^2 \log n)$. Roditty and Zwick [9] present a randomized algorithm with $\tilde{O}(m\sqrt{n})$ running time for unweighted, directed graphs. See also [2].

For planar directed graphs, an $O(n \log^3 n)$ time recursive algorithm is given in [3]. Klein, Mozes, and Weimann [6] show how recursion can be avoided and improve the time bound to $O(n \log^2 n)$ using linear space.

Our contribution is to improve the time bound of [6] for planar graphs by giving a linear space algorithm with $O(n \log n)$ running time. Our result is obtained by a relatively simple adaptation of the $O(n \log n)$ time multiple-source shortest path algorithm of Klein [5] to the RPP.

The organization of the paper is as follows. In Section 2, we give some definitions, introduce some notation, and present some basic results that will prove useful later on. A large part of this section is taken from [5]. In Section 3, we show how the RPP can be split into two simpler sub-problems. Before presenting our algorithm, we show how to efficiently solve a problem related to the RPP in Section 4. The ideas introduced here will be a stepping stone towards obtaining our main result. We then give the algorithm for the first sub-problem in Section 5 and bound its time and space requirements in Section 6. In Section 7, we present an efficient algorithm for the other sub-problem. Finally, we make some concluding remarks in Section 8.

## 2 Definitions, Notation, and Toolbox

Let $G = (V, E)$ be a graph with non-negative edge lengths. For an edge $e \in E$, we let $l_G(e)$ denote its length in $G$. For two vertices, $u, v \in V$, $d_G(u, v)$ is the length of a shortest path in $G$ from $u$ to $v$ w.r.t. $l_G$. If there is no such path, $d_G(u, v) = \infty$. We let $V_G$ resp. $E_G$ denote $V$ resp. $E$.

We can assume w.l.o.g. that all shortest paths considered are simple. For a simple path $P = v_1 \rightarrow \cdots \rightarrow v_m$ in a directed graph $G$, $l_G(P) = \sum_{i=1}^{m-1} l_G(v_i, v_{i+1})$ denotes its length and for $1 \leq i \leq j \leq m$, $P[v_i, v_j]$ is the subpath $v_i \to \cdots \to v_j$. We let $f_P$ be the flow in $G$ assigning values to edges and reverses of edges of $G$ as follows: for each edge $(u, v)$ of $P$, $f_P(u, v) = -f_P(v, u) = 1$ and for all other edges/reverses of edges, $f_P$ is zero.

Let $T$ be a spanning tree in a directed graph $G = (V, E)$ and let $T$ be rooted at a vertex $s$. For a vertex $v \in V$, $T[v]$ is the simple path from $s$ to $v$ in $T$. We say that an edge $(u, v) \in E$ is *relaxed (w.r.t. $T$)* if $d_T(s, u) + l_G(u, v) \geq d_T(s, v)$ and we say that $(u, v)$ is *unrelaxed (w.r.t. $T$)* otherwise. Observe that all edges of $E_T$ are relaxed. For an unrelaxed edge $(u, v)$, removing the edge in $T$ ending in $v$ and reconnecting $T$ by adding $(u, v)$ is called *relaxing* the edge $(u, v)$. It is well-known that if all edges of $E$ are relaxed w.r.t. $T$, $T$ is a shortest path tree in $G$ with source $s$.

Since $T$ is a spanning tree of $G$, the edges of $E \setminus E_T$ define a spanning tree $T^*$ in the dual of $G$ rooted at the external face of $G$ (see [5]) and $T^*$ contains all unrelaxed edges of $G$, where we identify each edge of $G$ with its corresponding edge in the dual of $G$. We call $T^*$ the *dual of $T$ (in $G$)*. For an edge $(u, v)$ in $T^*$, we define $l_{T^*}(u, v) = d_T(s, u) + l_G(u, v) - d_T(s, v)$. A *leafmost unrelaxed edge (in $G$ w.r.t. $T$)* is an unrelaxed edge in $G$ (and hence it belongs to $T^*$) w.r.t. $T$ none of whose proper descendant edges in $T^*$ are unrelaxed in $G$ w.r.t. $T$.

Given a plane directed graph $G = (V, E)$, let $G_\infty$ be a plane graph obtained by adding a vertex $v_\infty$ to the interior of the external face of $G$ and an edge from $v_\infty$ to each vertex on the external face of $G$. For edges $(u, v)$, $(v, x)$, and $(v, y)$ in $G_\infty$, we say that $(v, x)$ is *left (right) of $(v, y)$ w.r.t. $(u, v)$* if $(v, x)$ occurs strictly between $(v, y)$ and $(u, v)$ in counter-clockwise (clockwise) order.

Given an edge $(v, y)$ on a simple path $P$ in $G$, we say that an edge $(v, x)$ *emanates left (right) from $P$* if either there is an edge $(u, v)$ preceding $(v, y)$ on $P$ and $(v, x)$ is left (right) of $(v, y)$ w.r.t. $(u, v)$ or if $v$ is the first vertex of $P$ and belongs to the external face of $G$ and $(v, x)$ is left (right) of $(v, y)$ w.r.t. the edge from $v_\infty$ to $v$ in $G_\infty$.

Given another simple path $Q$ in $G$ and a vertex $u \in V_P \cap V_Q$, we say that *$Q$ leaves $P$ from the left (right) at $u$* if there is an edge $(u, v)$ of $Q$ starting in $u$ which emanates left (right) from $P$. And we say that *$Q$ enters $P$ from the left (right) at $u$* if there is an edge $(v, u)$ of $Q$ ending in $u$ such that the reverse edge $(u, v)$ emanates left (right) from $P$.

If both $P$ and $Q$ start in the same vertex $s$ and end in the same vertex $t$, we say that $Q$ is *left (right)* of $P$ if the edges of positive flow in $f_Q - f_P$ define counter-clockwise (clockwise) cycles only (this definition is by Weihe [10] and is specialized in [5]).

For two spanning trees $T_1$ and $T_2$ in $G$, $T_1$ is *left* of $T_2$ if for all $v \in V$, path $T_1[v]$ is left of $T_2[v]$. If $T$ is a shortest path tree in $G$ with source $s$, we call it a *right-most* shortest path tree if every other shortest path tree in $G$ with source $s$ is left of $T$.

An $s$-rooted spanning tree $T$ is *right-short* if the following holds for all $v \in V$: if $P$ is a simple path in $G$ from $s$ to $v$ that is right of $T[v]$ and $l_G(P) \leq l_G(T[v])$ then $P = T[v]$. A right-most shortest path tree is right-short [5].

We will need two dynamic tree data structures which represent and maintain, respectively, a rooted spanning tree $T$ and its dual $T^*$ and which support the following operations:

`replace`$(e, e')$**:** replaces edge $e$ by edge $e'$.

`sum`$(x)$**:** returns the sum of lengths of edges from the root to vertex $x$.

`find`$()$**:** returns a leafmost unrelaxed edge

`change`$(x, \Delta)$**:** for each edge $e$ on the path between $x$ and the root, the length of $e$ is increased by the real number $\Delta$ if $e$ points towards the root and decreased by $\Delta$ otherwise.

Top trees [1] support the above in logarithmic time per operation, see [5]. Note that the `change`-operation can be extended to subpaths of the path between $x$ and the root by applying the operation twice.

## 3 Simplifying the Problem

In the following, let $G = (V, E)$ be an $n$-vertex planar directed graph with non-negative edge lengths and let $P = (v_0 = s) \rightarrow v_1 \rightarrow \cdots \rightarrow v_{m-1} \rightarrow (v_m = t)$ be a shortest path in $G$ from a vertex $s$ to a vertex $t$. For $i = 1, \ldots, m$, let $e_i$ denote the edge $(v_{i-1}, v_i)$. By transforming $G$ if necessary, we may assume that $s$ belongs to the external face of $G$. Since we are only interested in shortest paths ending in $t$, we may assume that this vertex has no outgoing edges.

Let $e_i \in E_P$ and let us analyze the structure of a shortest path $Q$ in $G$ from $s$ to $t$ avoiding $e_i$. Since $P$ is a shortest path in $G$, $Q$ can be chosen such

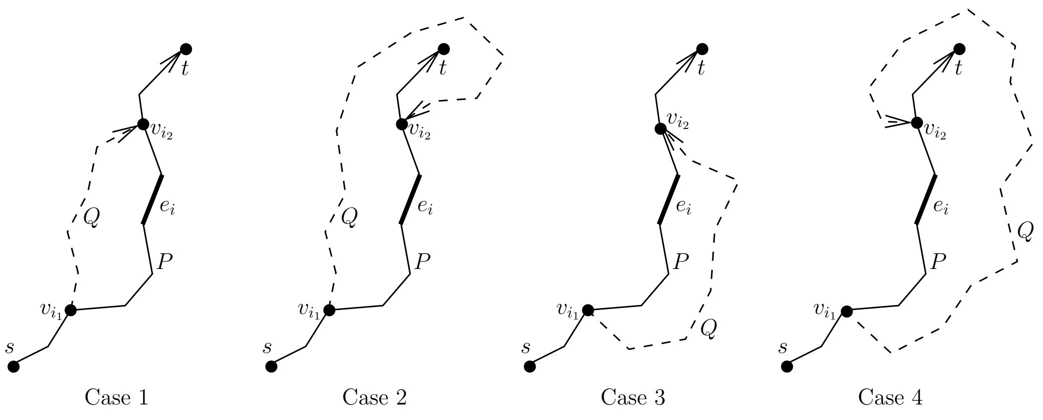

Figure 1: The four possible cases for shortest path $Q$.

that it has a decomposition $Q = Q_1Q_2Q_3$ where $Q_1 = P[s, v_{i_1}]$, $Q_3 = P[v_{i_2}, t]$ for some $0 \leq i_1 < i \leq i_2 \leq m$, and $Q_2$ is a path in $G$ from $v_{i_1}$ to $v_{i_2}$ containing no vertices of $P$ except $v_{i_1}$ and $v_{i_2}$.

There are now four possible cases (see Figure 1):

**Case 1:** $Q$ leaves $P$ from the left at $v_{i_1}$ and enters $P$ from the left at $v_{i_2}$

**Case 2:** $Q$ leaves $P$ from the left at $v_{i_1}$ and enters $P$ from the right at $v_{i_2}$

**Case 3:** $Q$ leaves $P$ from the right at $v_{i_1}$ and enters $P$ from the right at $v_{i_2}$

**Case 4:** $Q$ leaves $P$ from the right at $v_{i_1}$ and enters $P$ from the left at $v_{i_2}$

Our algorithm for the RPP consists of four phases where in phase $p$, $p = 1, 2, 3, 4$, shortest paths of the form $Q$ above are restricted to having the structure in case $p$. After these phases, we have four distance values for each edge $e_i$. The minimum of these four values is then the length of a shortest path in $G$ from $s$ to $t$ that avoids $e_i$.

In the following, we consider each phase separately. Due to symmetry, we may restrict our attention to phases 1 and 2. We start with phase 1 and consider phase 2 in Section 7.

We remove from $G$ edges $(u, v)$, $v \neq t$, for which either $(u, v)$ or $(v, u)$ emanates right from $P$ since these edges will not be needed in phase 1. Note that $G$ may contain more than one connected component. If so, we remove all components except the one containing $P$. Now, $P$ belongs to the external face of $G$ and is part of a counter-clockwise walk of that face.

By adding edges to interior faces of $G$ while keeping $G$ planar, we may assume that for each $v \in V$, there is a path in $G$ from $s$ to $v$ sharing no edges with $P$. We pick the lengths of these new edges sufficiently large so that finite shortest path distances will not decrease. With this modification of $G$, there is a shortest path tree in $G$ rooted at $s$ avoiding any given set of edges of $P$. Furthermore, we can ensure that these edges are avoided by increasing their lengths by a sufficiently large value ($M_+$ defined below). Note that $P$ remains on the external face of $G$ after these edges have been added.

Clearly, phase 1 corresponds to solving the RPP for the modified graph $G$. The idea is to use a dynamic tree data structure to maintain a shortest path tree in $G$ that initially avoids $e_m$, then $e_{m-1}$, then $e_{m-2}$, and so on until a shortest path tree avoiding $e_1$ is obtained. During this process, the distances in $G$ from $s$ to $t$ in the intermediate trees are computed. This is similar to the idea behind the multiple-source shortest path algorithm of Klein [5]. Indeed, we rely heavily on many of the results from that paper.

## 4 Solving a Related Problem

To simplify the presentation of our algorithm, we first consider a related problem. In this section, we show how to solve this problem in $O(n \log n)$ time. The ideas involved will prove useful in Sections 5, 6, and 7 where we present the RPP-algorithm.

The problem we consider is the following: for $i = 0, \ldots, m-1$, compute the length of a shortest path in $G$ from $s$ to $t$ avoiding every edge on $P[v_i, t]$. We call it the *Forbidden Suffix Paths Problem (FSPP)*.

In the following, let $M_+ < \infty$ be a value such that any simple path in $G$ has length strictly less than $M_+$. Pick, say, $M_+ = 1 + \sum_{e \in E} l_G(e)$.

We now present an $O(n \log n)$-time algorithm for the FSPP. Pseudo-code is given in Figure 2.

**Theorem 1.** *The algorithm in Figure 2 solves the FSPP for $G$ and can be implemented to run in $O(n \log n)$ time with $O(n)$ space requirement.*

*Proof.* Clearly, the algorithm solves the FSPP for $G$ by outputting the desired distances in reverse order.

We maintain $T$ and its dual $T^*$ using top trees supporting the operations of Section 2.

```
1. compute a rightmost shortest path tree $T$ in $G$ with source $s$
2. for $i = m, \ldots, 1$
3.    $l_G(e_i) := l_G(e_i) + M_+$
4.    while there exists an unrelaxed edge
5.       relax a leafmost unrelaxed edge
6.    output $d_T(s, t)$
```

Figure 2: The FSPP algorithm.

Note that edge lengths change during the course of the algorithm. Instead of explicitly making these changes in the underlying graph $G$, we choose an implementation maintaining the correct edge lengths in $T$ and its dual $T^*$. This works since $E_T \cup E_{T^*} = E$ so we still keep track of the lengths of all edges in $G$.

**Maintaining edge lengths in $T$:** in line 3, we increase $l_T(e_i)$ by $M_+$ if $e_i \in T$. Otherwise, we do nothing.

Now, suppose a new edge $(u, v)$ is about to be inserted into $T$ in line 5. The algorithm needs to compute $l_G(u, v)$, the length of the edge to be inserted into $T$. Edge $(u, v)$ is not already in $T$ so $(u, v) \in T^*$. Thus, its length in $T^*$ can be obtained in $O(\log n)$ time since $l_{T^*}(u, v) = |\texttt{sum}(v) - \texttt{sum}(u)|$ where the sum-operation is applied to $T^*$ . We can then use the following formula to determine $l_G(u, v)$ in $O(\log n)$ time:

$$l_G(u, v) = l_{T^*}(u, v) + d_T(s, v) - d_T(s, u) = l_{T^*}(u, v) + \texttt{sum}(v) - \texttt{sum}(u),$$

where the `sum`-operation is applied to $T$. We used the definition $l_{T^*}(u, v) = d_T(s, u) + l_G(u, v) - d_T(s, v)$.

**Maintaining edge lengths in $T^*$:** in line 3, either $e_i \in T$ or $e_i \notin T$. If $e_i \notin T$ then no distances from $s$ in $T$ change. Since $l_{T^*}(u, v) = d_T(s, u) + l_G(u, v) - d_T(s, v)$ for all edges $(u, v) \in T^*$, no edge lengths in $T^*$ change except for $l_{T^*}(e_i)$ which is increased by $M_+$. This update can be performed in $O(\log n)$ time with the `change`-operation.

If on the other hand $e_i \in T$ then $d_T(s, u)$ increases by $M_+$ for all vertices $u$ in the subtree of $T$ rooted at the vertex of $e_i$ furthest from $s$. For all other vertices $u$ of $T$, $d_T(s, u)$ does not change. Then observations from [5] give us that the edges of $T^*$ whose lengths change are all on the same path in $T^*$ and these lengths can be updated with the `change`-operation in $O(\log n)$ time.

When an edge is inserted into $T^*$ in line 5, we can compute its length in $O(\log n)$ time using ideas similar to those above for $T$.

**Bounding the number of relaxations:** we have now shown that each execution of line 3 can be executed in $O(\log n)$ time. As observed in [5], each relaxation can be performed within the same time bound and so can line 6 if we use the `sum`($t$)-operation on $T$.

What remains therefore is to give an $O(n)$ bound on the total number of relaxations. By [5], this reduces to showing that all intermediate trees constructed by the algorithm are right-short.

The initial tree is a rightmost shortest path tree and thus right-short. To see that line 3 preserves right-shortness, consider iteration $i$ and let $l_1$ resp. $l_2$ be length function $l_G$ just before resp. after line 3 is executed. Assume that $T$ is right-short w.r.t. $l_1$ and let $v \in V$ be given. We need to show that $l_2(Q) > l_2(T[v])$ for any simple path $Q \neq T[v]$ in $G$ from $s$ to $v$ that is right of $T[v]$.

So let $Q$ be such a path. Suppose first that $T[v]$ does not contain $e_i$. If $e_i \in Q$ then $l_2(Q) \geq M_+ > l_2(T[v])$ where the last inequality follows from the assumption that there is a path in $G$ from $s$ to $v$ avoiding every edge of $P$. So assume $e_i \notin Q$. Then $l_2(Q) = l_1(Q) > l_1(T[v]) = l_2(T[v])$ by right-shortness of $T$ w.r.t. $l_1$.

Now, suppose that $e_i \in T[v]$. Then also $e_i \in Q$ since $e_i$ is part of a counter-clockwise walk of the external face of $G$ and $Q$ is simple and right of $T[v]$. Then $l_2(Q) = l_1(Q)+M_+ > l_1(T[v])+M_+ = l_2(T[v])$ by right-shortness of $T$ w.r.t. $l_1$.

We conclude that line 3 preserves right-shortness. Results from [5] imply that lines 4 and 5 also preserve right-shortness. Hence, the total number of relaxations performed by the algorithm is $O(n)$ and the $O(n \log n)$ time bound follows. The bound on space also follows from [5]. □

## 5 The Algorithm

In this section, we present our algorithm for phase 1 of the RPP. Define graph $G' = (V', E')$ where $V' = V$ and $E'$ is obtained from $E$ by adding edge $e'_i = (v_{i-1}, t)$ of length $l_{G'}(e'_i) = d_G(v_{i-1}, t) + M_+$ for $i = 2, \ldots, m-1$. We

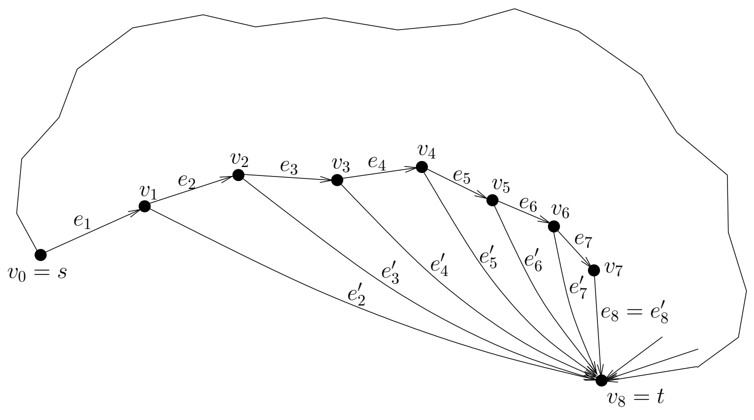

Figure 3: Graph $G'$ is obtained from $G$ by adding edges $e'_i$ for $i = 2, \ldots, m-1$, here shown for an instance with $m = 8$.

also define $e'_m = e_m$. Note that $G'$ is planar with new interior faces defined by triangles $e_i e'_i e'_{i+1}$ for $i = 2, \ldots, m-1$, see Figure 3. When convenient, we regard $G$ as a subgraph of $G'$.

Pseudo-code of our algorithm is shown in Figure 4. Notice the similarity with the FSPP algorithm in Figure 2.

**Theorem 2.** *The algorithm in Figure 4 solves the RPP for G.*

*Proof.* First, observe that in any iteration $i$, $l_{G'}(e_j) \geq M_+$ for $j = i, \ldots, m$ and $l_{G'}(e'_j) \geq M_+$ for $j = 2, \ldots, m$ when line 3 has just been executed. When line 6 is reached, $d_T(s, v_j)$ is thus the length of a shortest path in $G$ from $s$ to $v_j$ that avoids edges $e_i, \ldots, e_m$, for $j = 1, \ldots, m$. In particular, in lines 9 and 11, $d_T(s,t)$ is the length of a shortest path in $G$ from $s$ to $t$ that avoids edges $e_i, \ldots, e_m$.

Since $i + 1 > m$ if and only if we are in iteration $i = m$, the correct value for that iteration is thus output in line 11.

Now, assume that we are in iteration $i < m$ so that lines 6 to 10 are executed instead of line 11. In line 8, $l_{G'}(e'_j) = d_G(v_{j-1}, t) + M_+ - M_+ = l_G(P[v_{j-1}, t])$ for $j = i+1, \ldots, m$. Hence, $d_T(s, v_{j-1}) + l_{G'}(e'_j)$ is the length of a shortest path in $G$ from $s$ to $t$ that avoids edges $e_i, \ldots, e_{j-1}$ and uses edges $e_j, \ldots, e_m$.

There is a shortest path $Q$ in $G$ from $s$ to $t$ avoiding $e_i$ which can be decomposed into $Q_1 Q_2 Q_3$, where $Q_1 = P[s, v_{i_1}]$ and $Q_3 = P[v_{i_2}, t]$ for some $0 \leq i_1 < i \leq i_2 \leq m$, and where $Q_2$ is a shortest path in $G$ from $v_{i_1}$ to

```
1.  compute a rightmost shortest path tree $T$ in $G'$ with source $s$
2.  **for** $i = m, \ldots, 1$
3.    $l_{G'}(e_i) := l_{G'}(e_i) + M_+$
4.    **while** there exists an unrelaxed edge
5.      relax a leafmost unrelaxed edge
6.    **if** $i + 1 \leq m$
7.      **for** $j = i + 1, \ldots, m$, $l_{G'}(e'_j) := l_{G'}(e'_j) - M_+$
8.      compute $d = \min\{d_T(s, v_{j-1}) + l_{G'}(e'_j) | j = i + 1, \ldots, m\}$
9.      output $\min\{d_T(s, t), d\}$
10.     **for** $j = i + 1, \ldots, m$, $l_{G'}(e'_j) := l_{G'}(e'_j) + M_+$
11.   **else** output $d_T(s, t)$
```

Figure 4: The RPP algorithm for phase 1.

$v_{i_2}$ sharing no vertices with $P$ except $v_{i_1}$ and $v_{i_2}$. Combining this with the above observations, it follows that in line 9, $\min\{d_T(s, t), d\}$ is the length of a shortest path in $G$ from $s$ to $t$ that avoids $e_i$. This shows the correctness of the algorithm. □

## 6 Bounding Time and Space

We now give an implementation of the algorithm in Figure 4 and show that it has $O(n \log n)$ running time and $O(n)$ space requirement.

We maintain $T$, its dual $T^*$ (in $G'$), and their edge lengths (and not the edge lengths in $G'$ explicitly) as in the proof of Theorem 1. Since only edges of $G$ are relaxed in line 5 (all other edges of $G'$ have length at least $M_+$), we can use arguments similar to those in that proof to conclude that the total number of relaxations performed in line 5 is $O(n)$. Each execution of lines 9 and 11 takes $O(\log n)$ time with the `sum`-operation of Section 2.

We claim that each execution of lines 3, 7, and 10 also takes logarithmic time. From the proof of Theorem 1, this is true for line 3 so let us consider line 7 (line 10 is handled in a similar way).

Since $l_{G'}(e'_j) \geq M_+$ for $j = i + 1, \ldots, m$ when this line is reached, none of these edges belong to $T$ so no edge lengths in $T$ are affected in this line. It follows that $e'_{i+1}, \ldots, e'_m$ all belong to $T^*$ and they all need to decrease in length by $M_+$. We can make this update with a `change`-operation in $T^*$.

```
8.1.  remove from $T$ the edge $e$ ending in $t$ and reconnect by adding $e'_{i+1}$
8.2   for all edges $e' \in E \setminus \{e_m\}$ adjacent to $t$, $l_{G'}(e') := l_{G'}(e') + M_+$
8.3.  while there exists an unrelaxed edge
8.4.     relax a leafmost unrelaxed edge
8.5.  let $e'_j$ be the edge of $T$ ending in $t$
8.6.  let $d = d_T(s,t)$
8.7.  if $i + 1 \leq j - 1$
8.8.     for $j' = i+1, \ldots, j-1$, $l_{G'}(e'_{j'}) := l_{G'}(e'_{j'}) + M_+$
8.9   for all edges $e' \in E \setminus \{e_m\}$ adjacent to $t$, $l_{G'}(e') := l_{G'}(e') - M_+$
8.10. remove $e'_j$ from $T$ and reconnect by adding the edge $e$ from line 8.1
```

Figure 5: Sub-routine of the RPP algorithm computing the value $d$ in iteration $i$.

The above shows that if we ignore line 8, the algorithm can be implemented to run in $O(n \log n)$ time. In the following, we therefore focus on the problem of efficiently computing the value $d$.

The idea is to relax leafmost unrelaxed edges as in line 5 while ensuring that they all belong to $\{e'_{i+1}, \ldots, e'_m\}$. To guarantee that only $O(n)$ edges are relaxed throughout the course of the algorithm, we increase, in each iteration, the length of certain edges by $M_+$ so that they will not be relaxed again. We will show that these edges can be assumed not to belong to $T$ in subsequent iterations which ensures that the algorithm remains correct.

Line 8 is expanded to the sub-routine in Figure 5. We now prove its correctness.

Looking at lines 4 to 7 in Figure 4, we see that just before line 8.1 is executed, all edges of $E' \setminus \{e'_{i+1}, \ldots, e'_m\}$ are relaxed. Hence, just after the execution of this line, only edges adjacent to $t$ in $G'$ can be unrelaxed. Line 8.2 has the effect that no edges of $E \setminus \{e_m\}$ adjacent to $t$ are relaxed during the sub-routine. Combining this with the observation that in line 8.2, all edges in $\{e'_2, \ldots, e'_i\}$ have length at least $M_+$ whereas all edges in $\{e'_{i+1}, \ldots, e'_m\}$ have length strictly less than $M_+$ and $e'_{i+1} \in T$, it follows that when line 8.2 has just been executed, all unrelaxed edges of $G'$ belong to $\{e'_{i+2}, \ldots, e'_m\}$ during the while-loop in lines 8.3 and 8.4.

Line 8.10 therefore ensures that $T$ is the same at the beginning and end of the sub-routine. And line 8.9 ensures that this also holds for edge lengths of $E'$ except those changed in line 8.8.

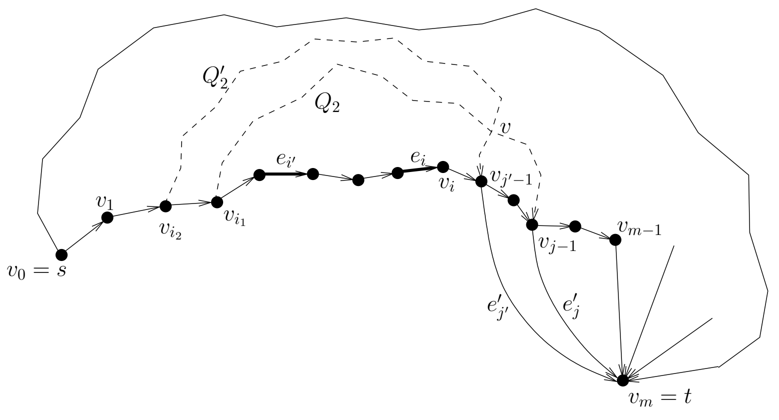

Figure 6: In the proof of Lemma 2, paths $Q_2$ and $Q'_2$ must share a vertex $v$.

The above observations imply that if line 8.8 is omitted, the correct value of $d$ is computed in line 8.6 since in that line, edges $\{e'_{i+1}, \ldots, e'_m\}$ are all relaxed w.r.t. $T$ so $d_T(s,t) = \min\{d_T(s, v_{j-1}) + l_{G'}(e'_j) | j = i+1, \ldots, m\}$. Theorem 2 then implies that the entire algorithm is correct. Lemma 2 below shows that this is also true if we include line 8.8. First, we need the following result.

**Lemma 1.** *Just before an edge $e'_j \in \{e'_{i+2}, \ldots, e'_m\}$ is relaxed in the subroutine in Figure 5, all edges in $\{e'_{i+1}, \ldots, e'_{j-1}\}$ are relaxed.*

*Proof.* Since initially, $e'_{i+1} \in T$ and since leafmost edges are relaxed, the lemma follows. □

**Lemma 2.** *Let $J = \{i+1, \ldots, j-1\}$ be the set of indices $j'$ in line 8.8 of iteration $i$ in Figures 4 and 5 and let $1 \le i' < i$. Let $G_i$ resp. $G_{i'}$ be the graph $G'$ just after line 7 has been executed in iteration $i$ resp. $i'$, but with the length of edge $e'_{j'}$ redefined as $d_G(v_{j'-1}, t)$ for all $j' \in J$. Then there is a shortest path in $G_{i'}$ from $s$ to $t$ avoiding each such edge.*

*Proof.* Let $Q'$ be a shortest path in $G_{i'}$ from $s$ to $t$ and suppose it contains $e'_{j'}$ for some $j' \in J$. Let $Q$ be the path from $s$ to $t$ in $T$ in line 8.5 of iteration $i$. Then $Q$ is a shortest path in $G_i$ from $s$ to $t$ avoiding $e_i$ and containing $e'_j$.

Since the restriction of $T$ to $E$ is right-short, $Q$ can be decomposed into $Q_1 Q_2$, where $Q_1$ is a subpath $P[s, v_{i_1}]$ of $P$ and $Q_2$ is a path from $v_{i_1}$ to $t$ containing no vertices of $P$ except $v_{i_1}$ and $v_{j-1}$ and containing $e'_j$ as the last edge, see Figure 6.

We may also assume that $Q'$ can be decomposed into $Q'_1 Q'_2$, where $Q'_1$ is a subpath $P[s, v_{i_2}]$ of $P$ and $Q'_2$ is a path from $v_{i_2}$ to $t$ containing no vertices of $P$ except $v_{i_2}$ and $v_{j'-1}$ and containing $e'_{j'}$ as the last edge.

We claim that $i_1 \geq i_2$, as shown in Figure 6. For suppose $i_1 < i_2$. Note that $i' < i$ and $i + 1 \leq j' < j$. When traversing $P$ from $s$ to $t$, we thus encounter $v_{i_1}$, $v_{i_2}$, $e_{i'}$, $e_i$, $v_{j'-1}$, and $v_{j-1}$ in that order. Hence, $Q$ and $Q'$ both avoid $e_i$ and $e_{i'}$ so these paths must be of equal length in $G_i$. We know that the algorithm relaxes $e'_j$ in line 8.4 of iteration $i$. By Lemma 1, just before this event occurs, $e'_{j'}$ must be relaxed. But since $l_{G_i}(Q) = l_{G_i}(Q')$, $e'_j$ must be relaxed as well at this point in time, a contradiction.

It follows that when traversing $P$ from $s$ to $t$, we encounter $v_{i_2}$, $v_{i_1}$, $e_i$, $v_{j'-1}$, and $v_{j-1}$ in that order. Due to planarity, this is only possible if $Q_2$ and $Q'_2$ share a vertex $v$, see Figure 6. Then $Q_2[v, t]$ and $Q'_2[v, t]$ have the same length in $G_{i'}$, implying that $Q'[s, v]Q[v, t]$ is a shortest path from $s$ to $t$ in $G_{i'}$. Since this path avoids $e'_{j'}$ for all $j' \in J$, the lemma follows. □

**Theorem 3.** *The algorithm in Figures 4 and 5 solves the RPP for $G$ and can be implemented to run in $O(n \log n)$ time using $O(n)$ space.*

*Proof.* We have already argued that the algorithm is correct when line 8.8 in Figure 5 is omitted. And Lemma 2 states that even if an edge in line 8.8 was not increased in length by $M_+$, the algorithm would not find a shorter path from $s$ to $t$ in subsequent iterations. This shows that the full algorithm is correct.

Proving the time bound is split into two parts: first, we ignore the time for relaxations and give an $O(n \log n)$ time bound for the remaining algorithm. Then we show that the total number of relaxations is $O(n)$. Since each relaxation can be performed in logarithmic time with our top tree data structure, the claim will follow.

We now consider the first part. If we exclude line 8, we have previously argued that the remaining lines can be executed in a total of $O(n \log n)$ time. So let us consider the sub-routine in Figure 5.

In line 8.1, we need to update the graph structure of $T$ and $T^*$ as well as edge lengths in these trees. The former can be accomplished with the `replace`-operation. As for the latter, we need to compute the length of the new edge $e'_{i+1}$ in $T$. This can be achieved in $O(\log n)$ time as in the proof of Theorem 1. Similarly, we can compute the length of the new edge $e$ in $T^*$ in $O(\log n)$ time.

For every other edge $(u, t)$ adjacent to $t$ in $G'$, $(u, t)$ belongs to $T^*$. Its old length (i.e., before the update) in $T^*$ is $d_T(s, u) + l_{G'}(u, t) - (d_T(s, v) + l_{G'}(e))$, where $e = (v, t)$. Its new length is $d_T(s, u) + l_{G'}(u, t) - (d_T(s, v_i) + l_{G'}(e'_{i+1}))$. Hence, the length of $(u, t)$ in $T^*$ should increase by $\Delta = d_T(s, v) + l_{G'}(e) - (d_T(s, v_i) + l_{G'}(e'_{i+1}))$.

Since $\Delta$ is independent of the choice of $(u, t)$, the lengths in $T^*$ of all edges adjacent to $t$ except $e$ and $e'_{i+1}$ should increase by $\Delta$. Since the set of these edges form at most three simple paths in $T^*$, a constant number of `change`-operations suffice to make this update.

We have shown that line 8.1 can be executed in $O(\log n)$ time. A similar argument shows the same bound for line 8.10.

Lines 8.2, 8.8, and 8.9 can also be executed in $O(\log n)$ time (since none of the edges in those three lines belong to $T$, the argument for line 7 applies). And line 8.6 also takes logarithmic time since $d_T(s, t)$ can be obtained as `sum`$(t)$ in $T$.

Now, let us focus on the second part of the proof: giving an $O(n)$ bound on the total number of relaxations. Previously, we showed this for line 5 so we only need to consider the relaxations performed in line 8.4.

We first observe that when the length of an edge is increased in line 8.8, it will never again drop below $M_+$.

Now, consider some iteration $i$ and suppose $k_i$ edges are relaxed in line 8.4. Since leafmost unrelaxed edges are relaxed and since the edge $e'_{i+1}$ initially belonging to $T$ has length $l_{G'}(e'_{i+1}) < M_+$, there must be at least $k_i$ edges in $\{e'_{i+2}, \ldots, e'_j\}$ of length strictly less than $M_+$ in line 8.5. Since the lengths of edges $e'_{i+2}, \ldots, e'_{j-1}$ are increased by $M_+$ in line 8.8, at least $k_i - 1$ of these lengths are increased from a value below to a value equal to or above $M_+$.

By the above observation, we can use a charging scheme to obtain the following bound on the total number of relaxations in line 8.4 over all $m$ iterations of the for-loop in lines 2 to 11:

$$\sum_{i=1}^{m} k_i = m + \sum_{i=1}^{m} (k_i - 1) \leq m + |\{e'_2, \ldots, e'_m\}| = 2m - 1 < 2n.$$

It follows that the algorithm can be implemented to run in $O(n \log n)$ time. Since the algorithm of Klein [5] has linear space requirement, it is easy to see that this bound also holds for our algorithm. □

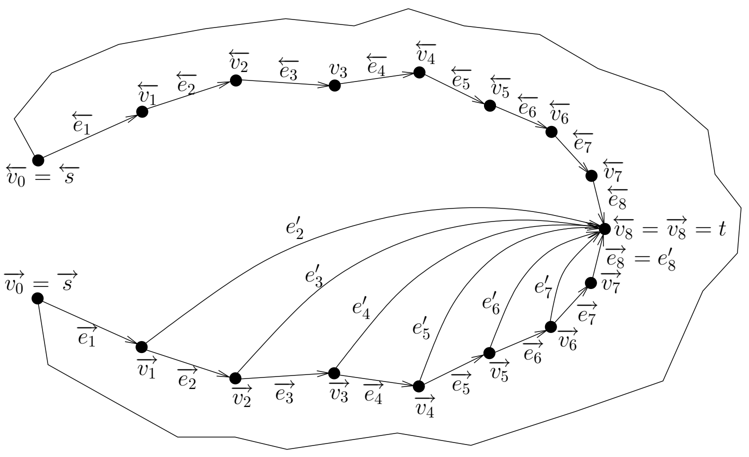

Figure 7: Graph $G'$, obtained from $G$ in phase 2 by adding edges $e'_i$ for $i = 2, \ldots, m-1$, here shown for an instance with $m = 8$.

## 7 Phase 2

Above, we showed how to solve the phase corresponding to case 1 in Section 3. We now consider phase 2, i.e. we restrict our attention to shortest paths $Q$ in $G$ from $s$ to $t$ avoiding an edge of $P$ with $Q$ leaving $P$ from the left and entering $P$ from the right, see Figure 1.

The algorithm for phase 2 is similar to that of Section 5. The main difference lies in the modification of $G$ and the construction of $G' = (V', E')$.

We modify $G$ essentially by making an incision in $G$ from $s$ to $t$ along $P$ (see Figure 7) and removing edges not needed in phase 2.

More formally, we start by removing path $P \setminus \{t\}$ and its incident edges. Then two copies, $\overleftarrow{P} = (\overleftarrow{v_0} = \overleftarrow{s}) \rightarrow \overleftarrow{v_1} \rightarrow \cdots \rightarrow (\overleftarrow{v_m} = t)$ and $\overrightarrow{P} = (\overrightarrow{v_0} = \overrightarrow{s}) \rightarrow \overrightarrow{v_1} \rightarrow \cdots \rightarrow (\overrightarrow{v_m} = t)$, of $P$ are inserted. These paths share only the last vertex $t$.

For $i = 0, \ldots, m-1$, we add to $G$ the edge $(\overleftarrow{v_i}, u)$ for each edge $(v_i, u)$ emanating left from $P$ at $v$ in the original graph $G$. Similarly, we add to $G$ the edge $(u, \overrightarrow{v_i})$ for each edge $(u, v_i)$ in $G$ entering $P$ from the right in the original graph $G$. The lengths of the inserted edges are identical to those in the original graph. Note that $\overleftarrow{P}$ and $\overrightarrow{P}$ belong to the external face of $G$.

As in phase 1, we add edges to interior faces of $G$ while keeping $G$ planar such that for each $v \in V$, there is a path in $G$ from $\overleftarrow{s}$ to $v$ sharing no edges with $\overleftarrow{P} \cup \overrightarrow{P}$. We pick the lengths of these new edges sufficiently large

```
1.  compute a rightmost shortest path tree $T$ in $G'$ with source $\overleftarrow{s}$
2.  **for** $i = m, \dots, 1$
3.    $l_{G'}(\overleftarrow{e_i}) := l_{G'}(\overleftarrow{e_i}) + M_+$
4.    **while** there exists an unrelaxed edge
5.      relax a leafmost unrelaxed edge
6.    **if** $i + 1 \leq m$
7.      **for** $j = i+1, \dots, m$, $l_{G'}(e'_j) := l_{G'}(e'_j) - M_+$
8.      compute $d = \min\{d_T(s, \overrightarrow{v_{j-1}}) + l_{G'}(e'_j) | j = i+1, \dots, m\}$
9.      output $\min\{d_T(s,t), d\}$
10.     **for** $j = i+1, \dots, m$, $l_{G'}(e'_j) := l_{G'}(e'_j) + M_+$
11.   **else** output $d_T(s,t)$
```

Figure 8: The RPP algorithm for phase 2.

so that finite shortest path distances will not decrease. We can perform this modification without having edges entering $\overleftarrow{P}$ or leaving $\overrightarrow{P}$. With this modification of $G$, there is a shortest path tree in $G'$ rooted at $\overleftarrow{s}$ avoiding any given set of edges of $\overleftarrow{P} \cup \overrightarrow{P}$. And we can ensure that these edges are avoided by increasing their lengths by $M_+$. Note that $\overleftarrow{P} \cup \overrightarrow{P}$ remains on the external face of $G$ after these edges have been added.

For $i = 1, \dots, m$, let $\overleftarrow{e_i} = (\overleftarrow{v_{i-1}}, \overleftarrow{v_i})$ and $\overrightarrow{e_i} = (\overrightarrow{v_{i-1}}, \overrightarrow{v_i})$. Phase 2 corresponds to solving the following problem on the modified graph $G$: for $i = 1, \dots, m$, compute the length of a shortest path in $G$ from $\overleftarrow{s}$ to $t$ that avoids edges $\overleftarrow{e_i}$ and $\overrightarrow{e_i}$. We will refer to this problem as the RPP for $G$.

Graph $G' = (V', E')$ is obtained from $G$ by adding, for $i = 2, \dots, m-1$, the edge $(\overrightarrow{v_i}, t)$ of length $d_G(v_i, t) + M_+$, where $M_+$ is defined as in Section 4. We let $e'_i$ denote this edge, see Figure 7. We also define $e'_m = \overrightarrow{e_m}$. Note that $G'$ is planar and of size $O(n)$. In $G'$, we set the length $l_{G'}(\overrightarrow{e_i})$ of $\overrightarrow{e_i}$ equal to $M_+$. Where appropriate, we regard $G$ as a subgraph of $G'$.

We now present the algorithm for phase 2. Pseudo-code is shown in Figure 8. Notice the similarity with the code in Figure 4.

**Theorem 4.** *The algorithm in Figure 8 solves the RPP for $G$.*

We omit the proof since it is almost identical to that of Theorem 2.

We expand line 8 to the sub-routine in Figure 9. Using ideas from Section 6, it follows that the entire algorithm is correct if line 8.8 is omitted. And Lemma 4 below shows that this also holds with this line included.

```
8.1.  remove from $T$ the edge $e$ ending in $t$ and reconnect by adding $e'_m$
8.2   **for** all edges $e' \in E \setminus \{\overrightarrow{e_m}\}$ adjacent to $t$, $l_{G'}(e') := l_{G'}(e') + M_+$
8.3.  **while** there exists an unrelaxed edge
8.4.      relax a leafmost unrelaxed edge
8.5.  let $e'_j$ be the edge of $T$ ending in $t$
8.6.  let $d = d_T(s,t)$
8.7.  **if** $j + 1 \leq m$
8.8.      **for** $j' = j+1, \ldots, m$, $l_{G'}(e'_{j'}) := l_{G'}(e'_{j'}) + M_+$
8.9   **for** all edges $e' \in E \setminus \{e_m\}$ adjacent to $t$, $l_{G'}(e') := l_{G'}(e') - M_+$
8.10. remove $e'_j$ from $T$ and reconnect by adding the edge $e$ from line 8.1
```

Figure 9: Sub-routine of the RPP algorithm for phase 2.

**Lemma 3.** *Just before an edge $e'_j \in \{e'_{i+1}, \ldots, e'_{m-1}\}$ is relaxed in the sub-routine in Figure 9, all edges in $\{e'_{j+1}, \ldots, e'_m\}$ are relaxed.*

*Proof.* Since initially, $e'_m \in T$ and since leafmost edges are relaxed, the lemma follows. □

**Lemma 4.** *Let $J = \{j+1, \ldots, m\}$ be the set of indices $j'$ in line* 8.8 *of iteration $i$ in Figures 8 and 9 and let $1 \leq i' < i$. Let $G_i$ resp. $G_{i'}$ be the graph $G'$ just after line 7 has been executed in iteration $i$ resp. $i'$, but with the length of edge $e'_{j'}$ redefined as $d_G(v_{j'-1}, t)$ for all $j' \in J$. Then there is a shortest path in $G_{i'}$ from $\overleftarrow{s}$ to $t$ avoiding each such edge.*

*Proof.* The proof is similar to that of Lemma 2.

Let $Q'$ be a shortest path in $G_{i'}$ from $\overleftarrow{s}$ to $t$ and suppose it contains $e'_{j'}$ for some $j' \in J$. Let $Q$ be the path from $\overleftarrow{s}$ to $t$ in $T$ in line 8.5 of iteration $i$. Then $Q$ is a shortest path in $G_i$ from $\overleftarrow{s}$ to $t$ avoiding $e_i$ and containing $e'_j$.

Since the restriction of $T$ to $E$ is right-short, $Q$ can be decomposed into $Q_1Q_2$, where $Q_1$ is a subpath $\overleftarrow{P}[\overleftarrow{s}, \overleftarrow{v_{i_1}}]$ of $\overleftarrow{P}$ and $Q_2$ is a path from $\overleftarrow{v_{i_1}}$ to $t$ containing no vertices of $\overleftarrow{P} \cup \overrightarrow{P}$ except $\overleftarrow{v_{i_1}}$ and $\overrightarrow{v_{j-1}}$ and containing $e'_j$ as the last edge, see Figure 10.

We may also assume that $Q'$ can be decomposed into $Q'_1Q'_2$, where $Q'_1$ is a subpath $\overleftarrow{P}[\overleftarrow{s}, \overleftarrow{v_{i_2}}]$ of $\overleftarrow{P}$ and $Q'_2$ is a path from $\overleftarrow{v_{i_2}}$ to $t$ containing no vertices of $\overleftarrow{P} \cup \overrightarrow{P}$ except $\overleftarrow{v_{i_2}}$ and $\overrightarrow{v_{j'-1}}$ and containing $e'_{j'}$ as the last edge.

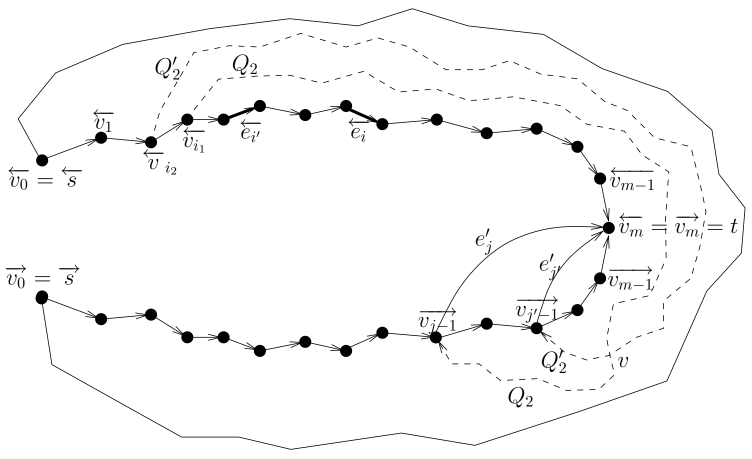
$Q_2'$
$Q_2$
$\overleftarrow{v_1}$
$\overleftarrow{v_{i_1}}$
$\overleftarrow{e_{i'}}$
$\overleftarrow{e_i}$
$\overleftarrow{v}_{i_2}$
$\overleftarrow{v_0} = \overleftarrow{s}$
$\overleftarrow{v_{m-1}}$
$\overleftarrow{v_m} = \overrightarrow{v_m} = t$
$e'_j$
$e'_{j'}$
$\overrightarrow{v_0} = \overrightarrow{s}$
$\overrightarrow{v_{m-1}}$
$\overrightarrow{v_{j-1}}$
$\overrightarrow{v_{j'-1}}$
$Q_2'$
$v$
$Q_2$

Figure 10: In the proof of Lemma 4, paths $Q_2$ and $Q_2'$ must share a vertex $v$.

We claim that $i_1 \geq i_2$, as shown in Figure 10. For suppose $i_1 < i_2$. Note that $i' < i$ and $j + 1 \leq j' \leq m$. When traversing $\overleftarrow{P}$ from $\overleftarrow{s}$ to $t$ followed by $\overrightarrow{P}$ backwards from $t$ to $\overrightarrow{s}$, we thus encounter $\overleftarrow{v_{i_1}}$, $\overleftarrow{v_{i_2}}$, $\overleftarrow{e_{i'}}$, $\overleftarrow{e_i}$, $\overrightarrow{v_{j'-1}}$, and $\overrightarrow{v_{j-1}}$ in that order. Hence, $Q$ and $Q'$ both avoid $\overleftarrow{e_i}$ and $\overleftarrow{e_{i'}}$ so these paths must be of equal length in $G_i$. We know that the algorithm relaxes $e'_j$ in line 8.4 of iteration $i$. By Lemma 3, just before this event occurs, $e'_{j'}$ must be relaxed. But since $l_{G_i}(Q) = l_{G_i}(Q')$, $e'_j$ must be relaxed as well at this point in time, a contradiction.

It follows that when traversing $\overleftarrow{P}$ from $\overleftarrow{s}$ to $t$ followed by $\overrightarrow{P}$ backwards from $t$ to $\overrightarrow{s}$, we encounter $\overleftarrow{v_{i_2}}$, $\overleftarrow{v_{i_1}}$, $\overleftarrow{e_i}$, $\overrightarrow{v_{j'-1}}$, and $\overrightarrow{v_{j-1}}$ in that order. Due to planarity, this is only possible if $Q_2$ and $Q_2'$ share a vertex $v$, see Figure 10. Then $Q_2[v,t]$ and $Q_2'[v,t]$ have the same length in $G_{i'}$, implying that $Q'[\overleftarrow{s}, v]Q[v,t]$ is a shortest path from $\overleftarrow{s}$ to $t$ in $G_{i'}$. Since this path avoids $e'_{j'}$ for all $j' \in J$, the lemma follows. $\square$

We can now bound the time and space used for phase 2.

**Theorem 5.** *The algorithm in Figure 4 and Figure 5 can be implemented to run in $O(n \log n)$ time using $O(n)$ space.*

*Proof.* Lemma 4 and observations above show the correctness of the algorithm. To prove an $O(n \log n)$ time bound, it suffices to show that the total number of relaxations performed in line 8.4 is $O(n)$.

We first observe that when the length of an edge is increased in line 8.8, it will never again drop below $M_+$.

Now, consider some iteration $i$ and suppose $k_i$ edges are relaxed in line 8.4. Since leafmost unrelaxed edges are relaxed, there must be at least $k_i$ edges in $\{e'_{j+1}, \ldots, e'_{m-1}\}$ of length strictly less than $M_+$ in line 8.5. Since the lengths of edges $e'_{j+1}, \ldots, e'_{m-1}$ are increased by $M_+$ in line 8.8, at least $k_i - 1$ of these lengths are increased from a value below to a value equal to or above $M_+$. The rest of the proof is identical to the proof of Theorem 3. $\square$

We have shown that phases 1 and 2 can be executed in $O(n \log n)$ time and $O(n)$ space. By symmetry, this also holds for phases 3 and 4. We can therefore conclude with the following result.

**Theorem 6.** *For a directed planar $n$-vertex graph with non-negative edge-lengths, the replacement paths problem can be solved in $O(n \log n)$ time with $O(n)$ space.*

## 8 Concluding Remarks

Given an $n$-vertex planar directed graph $G$ with non-negative edge lengths and given a shortest path $P$ in $G$ from a vertex $s$ to a vertex $t$, we presented a linear-space algorithm that computes, for each edge $e \in P$, the length of a shortest path in $G$ from $s$ to $t$ that avoids $e$. Running time is $O(n \log n)$, improving on a bound of $O(n \log^2 n)$ by Klein, Mozes, and Weimann.

## References

[1] S. Alstrup, J. Holm, K. de Lichtenberg, and M. Thorup. Maintaining information in fully-dynamic trees with top trees. ArXiv cs.DS/0310065.

[2] C. Demetrescu, M. Thorup, R. Alam Chaudhury, and V. Ramachandran. Oracles for distances avoiding a link-failure. Preliminary version of this paper appears in Proc. of 13th SODA, pages 838–843, 2002.

[3] Y. Emek, D. Peleg, and L. Roditty. A Near-Linear Time Algorithm for Computing Replacement Paths in Planar Directed Graphs. In SODA'08: Proceedings of the Nineteenth Annual ACM-SIAM symposium on Discrete Algorithms, pages 428–435, Philadelphia, PA, USA, 2008, Society for Industrial and Applied Mathematics.

[4] J. Hershberger, S. Suri, and A. Bhosle. On the difficulty of some shortest path problems. In Proc. of the 20th STACS, pages 343–354, 2003.

[5] P. N. Klein. Multiple-source shortest paths in planar graphs. Proceedings, 16th ACM-SIAM Symposium on Discrete Algorithms, 2005, pp. 146–155.

[6] P. N. Klein, S. Mozes, and O. Weimann. Shortest Paths in Directed Planar Graphs with Negative Lengths: a Linear-Space $O(n \log^2 n)$-Time Algorithm. Proc. 19th Ann. ACM-SIAM Symp. Discrete Algorithms, p. 236–245, 2009.

[7] K. Malik, A. K. Mittal, and S. K. Gupta. The $k$ most vital arcs in the shortest path problem. Operations Research Letters, 8(4):223–227, 1989.

[8] E. Nardelli, G. Proietti, and P. Widmayer. A faster computation of the most vital edge of a shortest path. Information Processing Letters, 79 (2): 81–85, 2001.

[9] L. Roditty and U. Zwick. Replacement paths and $k$ simple shortest paths in unweighted directed graphs. In Proc. Automata, Languages and Programming, 32nd International Colloquium, 249–260, 2005.

[10] K. Weihe. Maximum $(s,t)$-flows in planar networks in $O(|V| \log |V|)$ time. JCSS 55 (1997), pp. 454–476.