

A Near-Linear Time Algorithm for Computing Replacement Paths in Planar Directed Graphs

Yuval Emek *

David Peleg *

Liam Roditty *

Abstract

Let $G = (V(G), E(G))$ be a weighted directed graph and let P be a shortest path from s to t in G . In the *replacement paths* problem we are required to compute for every edge e in P , the length of a shortest path from s to t that avoids e . The fastest known algorithm for solving the problem in weighted directed graphs is the trivial one: each edge in P is removed from the graph in its turn and the distance from s to t in the modified graph is computed. The running time of this algorithm is $O(mn + n^2 \log n)$, where $n = |V(G)|$ and $m = |E(G)|$.

The replacement paths problem is strongly motivated by two different applications. First, the fastest algorithm to compute the k *simple shortest paths* from s to t in directed graphs [21, 13] repeatedly computes the replacement paths from s to t . Its running time is $O(kn(m + n \log n))$. Second, the computation of *Vickrey pricing* of edges in distributed networks can be reduced to the replacement paths problem. An open question raised by Nisan and Ronen [16] asks whether it is possible to compute the Vickrey pricing faster than the trivial algorithm described in the previous paragraph.

In this paper we present a near-linear time algorithm for computing replacement paths in weighted *planar* directed graphs. In particular, the algorithm computes the lengths of the replacement paths in $O(n \log^3 n)$ time. This result immediately improves the running time of the two applications mentioned above by almost a linear factor. Our algorithm is obtained by combining several new ideas with a data structure of Klein [12] that supports multi-source shortest paths queries in planar directed graphs in logarithmic time.

Our algorithm can be adapted to address the variant of the problem in which one is interested in the replacement path itself (rather than the length of the path). In that case the algorithm is executed in a preprocessing stage constructing a data structure that supports replacement path queries in time $\tilde{O}(h)$, where h is the number of hops in the replacement path. In addition, we can handle the variant in which vertices should be avoided instead of edges.

*Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel. E-mail: {yuval.emek,david.peleg,liam.roditty}@weizmann.ac.il. Supported in part by grants from the Israel Science Foundation and the Minerva Foundation.

1 Introduction

Computing the shortest paths in a graph is one of the most fundamental algorithmic problems in graph theory. Shortest paths are also used in numerous applications in communication networks. In certain real world networks links may occasionally fail and an alternative shortest path that does not use the failed link has to be found. More formally, consider a graph $G = (V(G), E(G))$ and two vertices $s, t \in V(G)$ and let P be a shortest path from s to t . In the *replacement paths* problem we are interested in computing, for every edge e on the path P , the length of a shortest path from s to t that avoids e .

The replacement paths problem for *undirected* graphs is well studied. Malik *et al.* [14] presents an algorithm for this problem whose running time is $O(m + n \log n)$. An $O(m\alpha(m, n))$ time algorithm is given for the problem by Nardelli *et al.* [15] in a stronger model of computation, using the linear time single source shortest paths algorithm of Thorup [19]. These results apply for undirected graphs only. This is partially explained by an $\Omega(m\sqrt{n})$ lower bound for the replacement paths problem on directed graphs established by Hershberger *et al.* [9] in the *path-comparison* model of Karger *et al.* [11].

In contrast, in *weighted* directed graphs no algorithm better than the trivial algorithm in which each edge of P is removed from the graph in its turn and a shortest paths tree is computed from s in the modified graph is known for the problem. This algorithm requires $O(mn + n^2 \log n)$ time. A randomized $\tilde{O}(m\sqrt{n})$ time algorithm for the replacement paths problem in unweighted directed graphs was recently presented by Roditty and Zwick [17]. Demetrescu *et al.* [2] presents an $O(mn^{1.5} + n^{2.5} \log n)$ algorithm that constructs a data structure supporting replacement path queries in constant time for any choice of source s and destination t . However, the computation of all pairs shortest paths is inherent in their algorithm and we are unaware of a method that significantly speeds up the running time of the algorithm if the source and destination are known in advance.

Beside the obvious motivation that the replacement paths problem has in itself, it is also strongly motivated as it is used in the following important applications. The first is the *k shortest paths problem*. Given a graph $G = (V(G), E(G))$, two vertices $s, t \in V(G)$ and an integer k , we are required to find the k shortest paths from s to t . Eppstein [4] presents an $O(m + n \log n + k)$ time algorithm for the directed version problem. However, the paths returned by Eppstein's algorithm are not necessarily simple, i.e., they may visit certain vertices more than once. In the *k shortest simple paths problem*, the paths returned should all be simple. Katoh *et al.* [10] present an $O(k(m + n \log n))$ time algorithm for solving the k shortest simple paths problem on undirected graphs. Yen [21] and Lawler [13] give an $O(kn(m + n \log n))$ time algorithms for solving the problem on weighted directed graphs. Their algorithm uses in each iteration a replacement paths algorithm. An $o(mn)$ algorithm for the replacement paths problem immediately implies an $o(kmn)$ algorithm for the k simple shortest paths problem. For unweighted directed graphs Roditty and Zwick [17] obtains a randomized algorithm whose running time is $\tilde{O}(km\sqrt{n})$. Recently, Roditty [18] presented an algorithm that approximates the k simple shortest paths in weighted directed graphs with a multiplicative stretch of $3/2$ in $\tilde{O}(km\sqrt{n})$ time.

Another application of the replacement paths problem comes from auction theory. The *Vickrey pricing* of links that are owned by selfish agents in a directed network is computed by solving the replacement paths problem. Let $G = (V(G), E(G))$ be a directed network, where links are owned by selfish agents and consider some $s, t \in V(G)$. Nisan and Ronen [16] show that if we like to find the shortest path from s to t , a mechanism that offers to pay $\delta_{G|\ell(e)=\infty}(s, t) - \delta_{G|\ell(e)=0}(s, t)$ to the owner of e for any link e on the shortest path from s to t and zero otherwise forces the links owners to

reveal their true costs. This kind of pricing is called *Vickrey pricing*. It is easy to see that computing the first value for every edge in the graph with respect to s and t is equivalent to computing the replacement paths from s to t . No algorithm other than the trivial one is known. (For more details see Hershberger and Suri [8] and Demetrescu *et al.* [2]). It was raised as an open question by Nisan and Ronen [16] whether it is possible to compute the Vickrey pricing faster.

Thus finding an algorithm for the replacement paths problem in weighted directed graphs whose running time is $o(mn + n^2 \log n)$ is an important open problem with many implications. Also, as we have mentioned before, in undirected graphs it is possible to compute the replacement paths in $O(m + n \log n)$ time. Explaining this huge gap between directed and undirected graphs is an interesting theoretical challenge.

Planar graphs have been the subject of many algorithmic studies, in part due to their special theoretical properties and in part due to their relevance to “real life” instances such as road networks. In particular, many problems involving distances and shortest paths attracted special interest in the context of planar graphs. Quoting Thorup [20]: “efficient tools for navigating in planar networks should be of inherent interest to any lazy creature living on a sphere like the surface of planet Earth”. The replacement paths problem admits strong relations to navigation in networks suffering from an occasional failed edge (or vertex), hence we believe, should be studied separately on planar graphs.

In this paper we present an algorithm that solves the replacement paths problem on weighted *planar* directed graphs in $O(n \log^3 n)$ time. This result immediately improves the running time of the two applications mentioned above by almost a linear factor on weighted planar directed graphs. Our algorithm is composed of several ideas. Each idea on its own is not sufficient for obtaining the improved algorithm, however when combined together in the right manner, these ideas yield a near-linear time algorithm. The main ingredient of our algorithm is a two layered recursive decomposition of tentative replacement paths that prunes many paths while incurring a low computational cost. Another ingredient of the algorithm is the multi-source shortest paths data structure of Klein [12]. Based on a simple observation we show that all the shortest paths information that is needed for the two layered recursive decomposition can be obtained by applying Klein’s preprocessing algorithm on a designated graph.

Our algorithm can be easily adapted to solve the replacement paths problem with respect to vertices, that is, for every vertex of the shortest path compute the length of a shortest path that avoids this vertex. It can also be adapted to handle the version of the problem in which one is interested in obtaining the replacement path rather than its length. This variant of the algorithm runs in time $O(n \log^3 n)$ during a preprocessing stage producing a data structure that supports replacement path queries in time $O(h \log \log n)$, where h is the number of hops in the replacement path. The details of these two variants are omitted from the extended abstract.

The rest of this paper is organized as follows. In the next section we define the notions used throughout the paper. Our replacement paths algorithm is presented in Section 3. The main procedure employed by the algorithm is described in Section 4. In Section 5 we present the adaptation of Klein’s data structure [12] to our needs.

2 Preliminaries

Let G be a graph. We denote the vertex set and edge set of G by $V(G)$ and $E(G)$, respectively. The edges of the graph may be assigned with non-negative *lengths* $\ell : E(G) \rightarrow \mathbb{R}_{\geq 0}$. Given a path

$P = (u_0, \dots, u_{p+1})$ in G , the length of P is defined to be $\text{len}(P) = \sum_{i=0}^p \ell(u_i, u_{i+1})$. The distance from vertex x to vertex y in G , denoted $\delta_G(x, y)$, is defined as the length of the shortest path from x to y in G . We refer to the prefix (respectively, suffix) of P that ends at u_i (resp., begins at u_j) as the u_i -*prefix* of P (resp., the u_j -*suffix* of P). If $j \leq i$ then the $[u_j, u_i]$ -*subpath* of P is merely the intersection of the u_i -prefix of P and the u_j -suffix of P . Throughout, we consider an n -vertex planar directed graph G with nonnegative edge lengths and some given source vertex $s \in V(G)$, destination vertex $t \in V(G)$, and fix a shortest path $P = (u_0, u_1, \dots, u_{p+1})$ from $s = u_0$ to $t = u_{p+1}$.

For an edge $e \in E(P)$, define $\text{replace}(s, t, e)$ to be the length of the shortest path from s to t in G that avoids the edge e . Such a shortest path, realizing $\text{replace}(s, t, e)$, is called a *replacement path* for e . The definition of $\text{replace}(s, t, e)$ can be extended to all edges in G by fixing $\text{replace}(s, t, e) = \delta_G(s, t)$ for every edge $e \in E(G) - E(P)$.

Let \bar{G} be the undirected graph that results from ignoring the orientation of the edges in G . Clearly, the planar embedding of G is suitable for \bar{G} . Let v be an arbitrary vertex in $V(\bar{G})$. Given some three edges $e_1, e_2, e_3 \in E(\bar{G})$ incident with v , we say that e_2 is *between* e_1 and e_3 if a counterclockwise traversal of the edges incident with v that begins at e_1 reaches e_2 before it reaches e_3 . Consider some simple path $Q = (v_0, \dots, v_{q+1})$ in \bar{G} . The last definition enables a classification of the edges incident with the internal vertices of Q in the following manner. Let v_j be an internal vertex in Q (i.e., $1 \leq j \leq q$) and let e be an edge incident with v_j such that $e \notin E(Q)$. We say that e is to the *left* of Q if e is between the edge (v_j, v_{j+1}) and the edge (v_j, v_{j-1}) , otherwise e is to the *right* of Q .

The notion of edges to the left of a path and to the right of a path is extended to directed graphs in a natural way: given a directed graph G , a (directed) path Q in G and some edge e incident with an internal vertex v in Q (e may be either incoming or outgoing with respect to v), we say that e is to the *left* (respectively, to the *right*) of Q if the unoriented image of e in \bar{G} is to the left (resp., to the right) of the unoriented image of Q . We sometimes refer to e simply as an *L-edge* (resp., *R-edge*).

3 The replacement paths algorithm

We present an $O(n \log^3 n)$ time algorithm that given an n -vertex planar directed graph G with nonnegative edge lengths, a source vertex $s \in V(G)$, a destination vertex $t \in V(G)$ and a shortest path $P = (u_0, u_1, \dots, u_{p+1})$ from $s = u_0$ to $t = u_{p+1}$, computes the values $\text{replace}(s, t, e)$ for all edges e along the shortest path P .

We refer to the vertices (respectively, edges) of P as *forbidden* vertices (resp., edges). Consider some simple path Q from s to t in G . We say that Q is *hunched* if Q can be rewritten as $Q = Q' \cdot \chi \cdot Q''$, where Q' is a (possibly empty) prefix of P , Q'' is a (possibly empty) suffix of P and χ is a nonempty path that avoids forbidden edges and vertices other than its endpoints. The segment χ is called a *detour* of P . The first and last vertices of χ are referred to as the *exit* and *entry* vertices of Q , respectively (refer to Figure 1 for illustration). It is well known (and easy to verify) that for every forbidden edge e , if $\text{replace}(s, t, e) < \infty$, then there exists a hunched path that realizes $\text{replace}(s, t, e)$ (in other words, if e can be replaced, then it can be replaced by a hunched path).

Our algorithm constructs a list \mathcal{L} of triplets of the form (i, j, λ) , where $0 \leq i < j \leq p + 1$ and λ is a nonnegative real number. The triplet (i, j, λ) in \mathcal{L} indicates that there exists a hunched path Q with exit vertex u_i and entry vertex u_j that satisfies $\text{len}(Q) = \lambda$. In particular, the path Q avoids the edges $(u_i, u_{i+1}), (u_{i+1}, u_{i+2}), \dots, (u_{j-1}, u_j)$. When the list \mathcal{L} is constructed, the algorithm processes

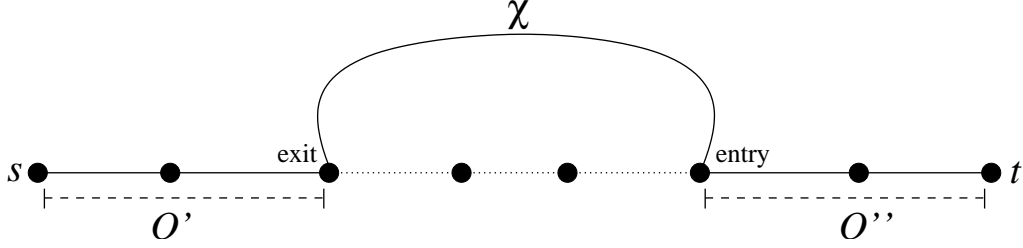


Figure 1: A hunched path. The forbidden vertices and edges along the dotted segment are avoided.

it to compute the values $replace(s, t, e)$ for all edges $e \in E(P)$.

Fix an arbitrary planar embedding of G . Recall that the non-forbidden edges incident with u_1, \dots, u_p in G are classified as L -edges and R -edges (see Section 2). Given some $d, d' \in \{L, R\}$ and a hunched path Q with detour χ , we say that Q is (d, d') -hunched (and that χ is a (d, d') -detour) if χ begins with a d -edge (outgoing from the exit vertex of Q) and ends with a d' -edge (incoming to the entry vertex of Q).

Let \tilde{G} be the directed graph obtained from G by removing the forbidden edges, namely, $V(\tilde{G}) = V(G)$ and $E(\tilde{G}) = E(G) - E(P)$. Our algorithm relies on a novel tool, referred to as the *Path Avoiding Distance Oracle (PADO)*, that enables fast computation of distances in \tilde{G} under some constraints. The PADO requires an $O(n \log n)$ time preprocessing stage. Upon completion of the preprocessing stage, the PADO supports queries of the following type in $O(\log n)$ time: given some $1 \leq i < j \leq p$ and $d, d' \in \{L, R\}$, compute the length λ of a shortest path ψ from u_i to u_j in \tilde{G} such that (1) ψ begins with a d -edge; (2) ψ ends with a d' -edge; and (3) ψ avoids forbidden vertices other than u_i and u_j , namely, $V(\psi) \cap V(P) = \{u_i, u_j\}$. (Observe that ψ avoids the forbidden edges regardless of (3) as these edges are not part of \tilde{G} .) If such a path ψ does not exist, then the query returns $\lambda = \infty$. We denote such a query by $PAD\text{-}query_{G,d,d'}(i, j)$. The PADO is described in Section 5.

Consider some $d, d' \in \{L, R\}$. Note that if $\lambda = PAD\text{-}query_{G,d,d'}(i, j) < \infty$, then there exists a (d, d') -hunched path Q in G with exit vertex u_i and entry vertex u_j of length $len(Q) = \delta_G(s, u_i) + \lambda + \delta_G(u_j, t)$. Given some $1 \leq i < j \leq p$, define

$$\widehat{len}_{d,d'}(i, j) = \delta_G(s, u_i) + PAD\text{-}query_{G,d,d'}(i, j) + \delta_G(u_j, t)$$

as the length of the shortest (d, d') -hunched path with exit vertex u_i and entry vertex u_j . (If such a path does not exist, then $\widehat{len}_{d,d'}(i, j) = \infty$.) By employing the PADO, we can compute the value $\widehat{len}_{d,d'}(i, j)$ in time $O(\log n)$.

Fix some $d, d' \in \{L, R\}$. The list \mathcal{L} is constructed recursively in a divide and conquer fashion. The input to every recursive invocation is a subpath of P , where (u_1, \dots, u_p) is the input to the first invocation. Consider the subpath (u_a, \dots, u_b) input to some recursive invocation and let $\sigma = b - a > 0$ and $\mu = \lceil (a + b) / 2 \rceil$. To simplify the presentation of our algorithm, we first append to \mathcal{L} the triplets $(i, \mu, \widehat{len}_{d,d'}(i, \mu))$ and $(\mu, j, \widehat{len}_{d,d'}(\mu, j))$ for all $a \leq i < \mu$ and $\mu < j \leq b$. This is performed in time $O(\sigma \log n)$. Next, we employ Procedure **District** (described in Section 4) that computes the indices $f(i)$ for all $a \leq i < \mu$ and $g(j)$ for all $\mu < j \leq b$, where

$$f(i) = \operatorname{argmin}_{\mu < j \leq b} \{\widehat{len}_{d,d'}(i, j)\} \quad g(j) = \operatorname{argmin}_{a \leq i < \mu} \{\widehat{len}_{d,d'}(i, j)\},$$

in time $O(\sigma \log \sigma \log n)$. The corresponding triplets $\left\{ \left(i, f(i), \widehat{\text{len}}_{d,d'}(i, f(i)) \right) \mid a \leq i < \mu \right\}$ and $\left\{ \left(g(j), j, \widehat{\text{len}}_{d,d'}(g(j), j) \right) \mid \mu < j \leq b \right\}$ are appended to \mathcal{L} . We then proceed recursively with the subpaths $(u_a, \dots, u_{\mu-1})$ and $(u_{\mu+1}, \dots, u_b)$. The above (recursive) process is repeated for the four possible choices of $d, d' \in \{L, R\}$. The running time of the divide and conquer stage of our algorithm obeys the recurrence $T(p) \leq O(p \log p \log n) + 2T(p/2) = O(p \log^2 p \log n)$.

Upon completion of the recursive algorithm, we employ Dijkstra's single source shortest path algorithm [1, 3] to compute the distances from s and the distances to t in \widetilde{G} (that is, G without the forbidden edges) and append to \mathcal{L} the triplets $(0, i, \delta_{\widetilde{G}}(s, u_i) + \delta_G(u_i, t))$ and $(j, p+1, \delta_G(s, u_j) + \delta_{\widetilde{G}}(u_j, t))$ for all $1 \leq i \leq p+1$ and $0 \leq j \leq p$. This stage requires an additional $O(n \log n)$ time.

Finally, the contents of the list \mathcal{L} are examined in order to compute the values $\text{replace}(s, t, e)$ for all edges $e \in E(P)$. The path P is traversed from beginning to end, processing the edges (u_i, u_{i+1}) for $i = 0, 1, \dots, p$. While traversing P , we maintain a heap \mathcal{H} of triplets from \mathcal{L} . The heap \mathcal{H} is designed to contain the triplets $\{(k, k', \lambda) \in \mathcal{L} \mid k \leq i < k'\}$ when processing the edge $(u_i, u_{i+1}) \in E(P)$. Therefore we can assign $\text{replace}(s, t, (u_i, u_{i+1})) \leftarrow \min \{\lambda \mid (k, k', \lambda) \in \mathcal{H}\}$ for every $0 \leq i \leq p$. The heap is maintained as follows. When the traversal of P reaches the edge (u_i, u_{i+1}) , all triplets of the form (i, j, λ) are inserted into \mathcal{H} and all triplets of the form (j, i, λ) are removed from \mathcal{H} . Note that $O(p)$ triplets are appended to \mathcal{L} on every level of the recursion, which sums up to $O(p \log p)$ triplets altogether. As every triplet is inserted into (and removed from) \mathcal{H} exactly once, the heap can be maintained in time $O(p \log^2 p)$.

We now turn to analyze our algorithm. By the construction of the list \mathcal{L} , the existence of the triplet (i, j, λ) in \mathcal{L} indicates that there exists a hunched path in G of length λ with exit vertex u_i and entry vertex u_j . It remains to prove that for every edge $e = (u_k, u_{k+1}) \in E(P)$, if $\text{replace}(s, t, e) = \lambda$, then upon termination of our algorithm, \mathcal{L} contains a triplet of the form (i, j, λ) for some $0 \leq i \leq k$ and $k+1 \leq j \leq p+1$. The case where $\text{replace}(s, t, e)$ is realized by a hunched path with exit vertex u_0 (respectively, entry vertex u_{p+1}) is trivial to analyze as \mathcal{L} contains the shortest hunched path with exit vertex u_0 (resp., u_j) and entry vertex u_i (resp., u_{p+1}) for every $1 \leq i \leq p+1$ (resp., $0 \leq j \leq p$).

In what follows we assume that there exists a (d, d') -hunched path Q of length $\text{len}(Q) = \lambda$ with exit vertex u_i and entry vertex u_j such that $1 \leq i \leq k$ and $k+1 \leq j \leq p$. Consider the deepest recursive invocation of our algorithm on a subpath $S = (u_a, \dots, u_b)$ such that $a \leq i < j \leq b$. Let $\mu = \lceil (a+b)/2 \rceil$. The subpaths $(u_a, \dots, u_{\mu-1})$ and $(u_{\mu+1}, \dots, u_b)$ are input to the subsequent recursive invocations, therefore $i \leq \mu \leq j$ as otherwise, the current recursive invocation is not the deepest. If $i = \mu$ or $j = \mu$, then we are done since \mathcal{L} contains the triplets $(i', \mu, \widehat{\text{len}}_{d,d'}(i', \mu))$ and $(\mu, j', \widehat{\text{len}}_{d,d'}(\mu, j'))$ for all $a \leq i' < \mu$ and $\mu < j' \leq b$. So suppose that $i < \mu < j$.

Assume without loss of generality that $k < \mu$, that is, the edge e appears on the first half of S . (The case where $k > \mu$ is proved by the same line of arguments, replacing i with j and $f(i)$ with $g(j)$.) Recall that upon completion of Procedure `District`, the list \mathcal{L} contains the triplet $(i, f(i), \widehat{\text{len}}_{d,d'}(i, f(i)))$. Since j is a candidate for being $f(i)$, we conclude that $\widehat{\text{len}}_{d,d'}(i, f(i)) \leq \lambda$. The assertion follows as $f(i) > \mu \geq k+1$.

The running time of our algorithm is $O(n \log n + p \log^2 p \log n)$, where the first term is due to the preprocessing of the PADO and the computation of distances from s and distances to t in G and \widetilde{G} . Since $p < n$, we may bound the running time by $O(n \log^3 n)$ (the algorithm is asymptotically faster if $p = o(n)$).

Theorem 3.1. *There is an $O(n \log^3 n)$ time algorithm that given an n -vertex planar directed graph*

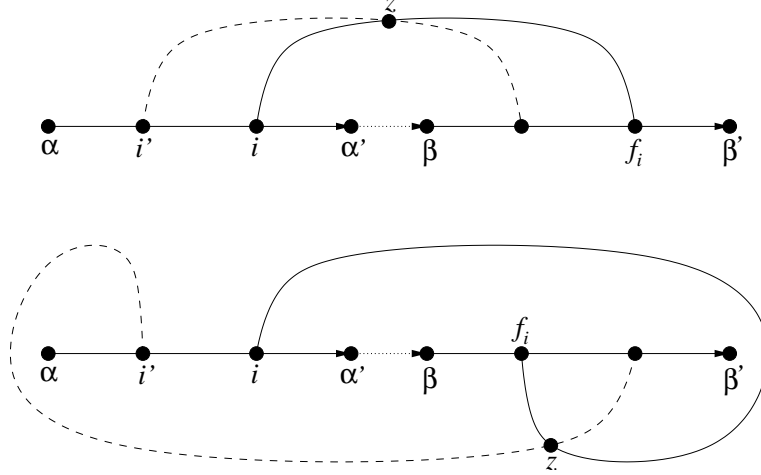


Figure 2: (L, L) -detours (top) and (L, R) -detours (bottom). The hunched path defined by the dashed detour cannot be strictly shorter than the hunched path defined by the detour that goes from $u_{i'}$ to u_{f_i} via z .

G with nonnegative edge lengths, a source vertex $s \in V(G)$, and a destination vertex $t \in V(G)$, computes the values $\text{replace}(s, t, e)$ for all edges $e \in E(G)$.

4 Procedure District

Let S be the $[u_a, u_b]$ -subpath of P , where $1 \leq a < b \leq p$. Let $\sigma = b - a$ and $\mu = \lceil (a + b)/2 \rceil$. Fixing some $d, d' \in \{L, R\}$, Procedure **District** computes the indices $f(i) = \text{argmin}_{\mu < j \leq b} \{\widehat{\text{len}}_{d, d'}(i, j)\}$ for all $a \leq i < \mu$ and the indices $g(j) = \text{argmin}_{a \leq i < \mu} \{\widehat{\text{len}}_{d, d'}(i, j)\}$ for all $\mu < j \leq b$ in time $O(\sigma \log \sigma \log n)$. Note that although $f(i)$ and $g(j)$ are determined by some global properties of the graph, the running time of Procedure **District** does not depend on the size of the graph (up to a logarithmic factor) but rather on the size of the current subpath. This is a key ingredient in the analysis of our algorithm.

We exploit a property of planar graphs, referred to by Fakcharoenphol and Rao [7] as the *non-crossing condition*. Consider some $a \leq i < \mu$ (respectively, $\mu < j \leq b$) and suppose that we have already computed the index $f(i)$ (resp., $g(j)$). Given some $a \leq i' < \mu$ (resp., $\mu < j' \leq b$), the non-crossing condition allows us to narrow the interval in which we search for the index $f(i')$ (resp., $g(j')$).

Consider the intervals $[\alpha, \alpha']$ and $[\beta, \beta']$ for some $a \leq \alpha < \alpha' < \mu < \beta < \beta' \leq b$. Define $f^{\beta, \beta'}(i) = \text{argmin}_{j \in [\beta, \beta']} \{\widehat{\text{len}}_{d, d'}(i, j)\}$ and $g^{\alpha, \alpha'}(j) = \text{argmin}_{i \in [\alpha, \alpha']} \{\widehat{\text{len}}_{d, d'}(i, j)\}$. The following lemma extends the ideas presented in [7] to (d, d') -hunched paths (refer to Figure 2 for illustration).

Lemma 4.1. *Let i and j be some indices in the intervals $[\alpha, \alpha']$ and $[\beta, \beta']$, respectively, and suppose that the indices $f_i = f^{\beta, \beta'}(i)$ and $g_j = g^{\alpha, \alpha'}(j)$ have already been computed. Then the computation of $f^{\beta, \beta'}(i')$ for all $i' \in [\alpha, i]$ (respectively, for all $i' \in [i, \alpha']$) can be restricted as follows:*

- (1) *If $d = d'$, then $f^{\beta, \beta'}(i')$ is in the interval $[f_i, \beta']$ (resp., $[\beta, f_i]$).*

(2) If $d \neq d'$, then $f^{\beta, \beta'}(i')$ is in the interval $[\beta, f_i]$ (resp., $[f_i, \beta']$).

Similarly, the computation of $g^{\alpha, \alpha'}(j')$ for all $j' \in [j, \beta']$ (respectively, for all $j' \in [\beta, j]$) can be restricted as follows:

(3) If $d = d'$, then $g^{\alpha, \alpha'}(j')$ is in the interval $[\alpha, g_j]$ (resp., $[g_j, \alpha']$).

(4) If $d \neq d'$, then $g^{\alpha, \alpha'}(j')$ is in the interval $[g_j, \alpha']$ (resp., $[\alpha, g_j]$).

Proof (Sketch). We prove the first claim (the other three are analogous). Fix some $d = d' \in \{L, R\}$. Let Q be the (d, d') -hunched path that realizes $\widehat{\text{len}}_{d, d'}(i, f_i)$ and let χ be Q 's detour. Let i' be some index in $[\alpha, i]$ (respectively, in $[i, \alpha']$) and let k be some index in $[\beta, f_i]$ (resp., in $(f_i, \beta']$). Consider the (d, d') -hunched path R that realizes $\widehat{\text{len}}_{d, d'}(i', k)$ and let ψ be R 's detour. We argue that $\widehat{\text{len}}_{d, d'}(i', f_i) \leq \text{len}(R)$ which establishes the claim.

Topology-wise, we have to consider two different types of the (d, d') -detours χ and ψ determined by whether or not the vertex s is internal to the shape enclosed by the detour and the corresponding subpath of P . The key observation is that due to the planarity of the graph, in all cases the detours χ and ψ must intersect at some vertex $z \in V(G)$ (see Figure 2). By the definition of f_i , the z -suffix of Q is not longer than the z -suffix of R . Therefore the (d, d') -hunched path defined by the detour that goes from $u_{i'}$ to z along ψ and from z to f_i along χ is not longer than R . \square

We describe the computation of the indices $f(i)$ for all $a \leq i < \mu$. The computation of the indices $g(j)$ for all $\mu < j \leq b$ is analogous and performed separately. Procedure `District` works recursively on intervals $[\alpha, \alpha']$ and $[\beta, \beta']$ such that $a \leq \alpha < \alpha' < \mu < \beta < \beta' \leq b$. Initialize $\alpha = a$, $\alpha' = \mu - 1$, $\beta = \mu + 1$ and $\beta' = b$, and fix $i = \lceil (\alpha' - \alpha)/2 \rceil$. We first search for the index $f(i) = f^{\beta, \beta'}(i)$ in the interval $[\beta, \beta']$ in a straightforward manner by comparing $\widehat{\text{len}}_{d, d'}(i, j)$ for every $j \in [\beta, \beta']$. Employing the PADO, this step takes $O(\sigma \log n)$ time.

Lemma 4.1 guarantees that the process of computing the indices $f(i')$ for all $i' \in [\alpha, i]$ can be separated from the process of computing the indices $f(i')$ for all $i' \in (i, \alpha']$. One of these processes continues recursively in the interval $[\beta, f(i)]$ while the other continues recursively in the interval $[f(i), \beta']$. By the choice of i , the depth of the recursion is $O(\log \sigma)$, thus every index $j \in (\mu, b]$ participates in at most $O(\log \sigma)$ recursive invocations of the procedure. Consequently, the running-time of Procedure `District` is $O(\sigma \log \sigma \log n)$.

5 The Path Avoiding Distance Oracle (PADO)

Given an n -vertex planar directed graph G with nonnegative edge lengths, a designated planar embedding of G and a simple path $P = (u_0, u_1, \dots, u_{p+1})$ in G , the PADO requires an $O(n \log n)$ preprocessing stage, after which it answers queries of the type `PAD-query` $_{G, d, d'}(i, j)$ (see Section 3) in time $O(\log n)$ for every $d, d' \in \{L, R\}$ and $1 \leq i < j \leq p$. The main ingredient in the construction of the PADO is an algorithm presented by Klein [12]. On an n -vertex planar directed graph H with nonnegative edge lengths and a designated face ϕ in some planar embedding of H , the algorithm of [12] works in time $O(n \log n)$, constructing a data structure that supports queries of the following type in time $O(\log n)$: given an arbitrary vertex v in $V(H)$ and a vertex w on the boundary of ϕ , compute the distance $\delta_H(v, w)$. We denote such a query by `D-query` $_H(v, w)$.

Fix $\Delta = n \cdot \max\{\ell(e) \mid e \in E(G)\}$. (Observe that Δ is strictly greater than the length of any simple path in G and can be computed in linear time.) We describe a linear time construction that transforms G into a planar directed graph H with nonnegative edge lengths, referred to as a *faced graph*, that satisfies

- (1) $V(H) = (V(G) - \{u_0, \dots, u_{p+1}\}) \cup U_L \cup U_R$, where $U_L = \{u_{i,L} \mid 0 \leq i \leq p+1\}$ and $U_R = \{u_{i,R} \mid 0 \leq i \leq p+1\}$;
- (2) the vertices in $U_L \cup U_R$ are on the boundary of some face ϕ in (a designated planar embedding of) H ; and
- (3) for every $d, d' \in \{L, R\}$ and $1 \leq i < j \leq p$, if $\delta_H(u_{i,d}, u_{j,d'}) < 2\Delta$, then $\text{PAD-query}_{G,d,d'}(i, j) = \delta_H(u_{i,d}, u_{j,d'}) - \Delta$; if $\delta_H(u_{i,d}, u_{j,d'}) \geq 2\Delta$, then $\text{PAD-query}_{G,d,d'}(i, j) = \infty$.

When the construction of H (to be described soon) is completed, we feed the algorithm of [12] with H and ϕ . Subsequently, we can employ the resulting data structure to answer our queries.

The construction works as follows. First, remove from G all edges incident with u_0 or u_{p+1} except from the edges (u_0, u_1) and (u_p, u_{p+1}) . Let G' be the resulting directed graph. Next, transform G' into the directed graph G'' by removing the forbidden edges (the edges of P) and replacing the vertex u_i with the vertices $u_{i,L}$ and $u_{i,R}$ for every $0 \leq i \leq p+1$. An L -edge (respectively, R -edge) (u_i, v) in G' is replaced with the edge $(u_{i,L}, v)$ (resp., $(u_{i,R}, v)$) in G'' (keeping the same edge length) for every $1 \leq i \leq p$ and $v \in V(G') - V(P)$. An L -edge (respectively, R -edge) (v, u_i) of length ℓ in G' is replaced with the edge $(v, u_{i,L})$ (resp., $(v, u_{i,R})$) of length $\ell + \Delta$ in G'' for every $1 \leq i \leq p$ and $v \in V(G') - V(P)$. (In other words, the length of the outgoing edges remains unchanged, while an additive Δ term is added to the length of the incoming edges.) Finally, the directed graph H is constructed by adding the edges $\{(u_{0,L}, u_{0,R}), (u_{p+1,L}, u_{p+1,R})\}$ and $\{(u_{i,L}, u_{i+1,L}), (u_{i,R}, u_{i+1,R}) \mid 0 \leq i < p+1\}$, all of them of length 2Δ , to G'' (refer to Figure 3 for illustration). Note that an edge e incident with the vertex $u \in U_L \cup U_R$ in H is of length 2Δ if e connects u with another vertex in $U_L \cup U_R$. Otherwise, $\ell(e) < \Delta$ if e is outgoing from u and $\Delta \leq \ell(e) < 2\Delta$ if e is incoming to u .

We now turn to prove that H satisfies the three desired properties of a faced graph.

Lemma 5.1. *H is a faced graph.*

Proof. We begin with arguing that for every $d, d' \in \{L, R\}$ and $1 \leq i < j \leq p$, the distances in H satisfy $\text{PAD-query}_{G,d,d'}(i, j) = \delta_H(u_{i,d}, u_{j,d'}) - \Delta$ if $\delta_H(u_{i,d}, u_{j,d'}) < 2\Delta$ and $\text{PAD-query}_{G,d,d'}(i, j) = \infty$ if $\delta_H(u_{i,d}, u_{j,d'}) \geq 2\Delta$. First note that a path of length λ that realizes $\text{PAD-query}_{G,d,d'}(i, j)$ (assuming that $\text{PAD-query}_{G,d,d'}(i, j) < \infty$) corresponds to a path of length $\lambda + \Delta$ from $u_{i,d}$ to $u_{j,d'}$ in H , hence $\text{PAD-query}_{G,d,d'}(i, j) \geq \delta_H(u_{i,d}, u_{j,d'}) - \Delta$. Therefore if $\delta_H(u_{i,d}, u_{j,d'}) \geq 2\Delta$, then $\text{PAD-query}_{G,d,d'}(i, j) \geq \Delta$, which means that $\text{PAD-query}_{G,d,d'}(i, j) = \infty$ since Δ is strictly larger than the length of any simple path in G .

Assume that $\delta_H(u_{i,d}, u_{j,d'}) = \lambda < 2\Delta$. To see that $\text{PAD-query}_{G,d,d'}(i, j) \leq \lambda - \Delta$, consider a path ψ in H that realizes $\delta_H(u_{i,d}, u_{j,d'})$. The path ψ cannot contain any edge in $(U_L \cup U_R) \times (U_L \cup U_R)$ as the length of every such edge is 2Δ . If $V(\psi) \cap (U_L \cup U_R)$ contains any vertex u other than $u_{i,d}$ and $u_{j,d'}$, then ψ must contain two different edges of length at least Δ (one incoming to u and the other incoming to $u_{j,d'}$) and we must have $\text{len}(\psi) \geq 2\Delta$. By the construction of H , ψ corresponds to a path of length $\lambda - \Delta$ in G that should be considered as a candidate to realize $\text{PAD-query}_{G,d,d'}(i, j)$. The argument follows.

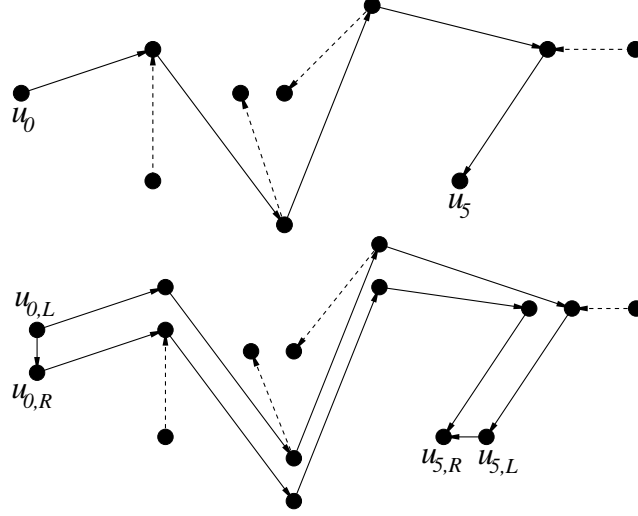


Figure 3: Top: the path P (solid lines) with some left and right edges (dashed lines). Bottom: the resulting faced graph.

It remains to argue that H is planar and that the vertices in $U_L \cup U_R$ are on the boundary of some face ϕ in H . Consider the (designated) planar embedding $\eta : V(G) \rightarrow \mathbb{R}^2$ that maps every vertex in G to a point in the Euclidean plane. Given a point $r = (r_x, r_y)$ in the plane, we denote the *magnitude* of r by $|r| = \sqrt{r_x^2 + r_y^2}$ and the *polar angle* of r by $\theta(r) = \arctan \frac{r_y}{r_x}$. Assume without loss of generality that η is a straight line planar embedding (see Fáry and de Fraysseix *et al.* [5, 6]). We design a straight line planar embedding $\eta' : V(H) \rightarrow \mathbb{R}^2$ for H that can be computed from η in linear time.

Clearly, η is a suitable (straight line) planar embedding for G' obtained from G by (possibly) removing some edges. When transforming G' into H , we replaced the vertex u_i with the vertices $u_{i,L}$ and $u_{i,R}$ for every $0 \leq i \leq p+1$ and the edge (u_i, u_{i+1}) with the edges $(u_{i,L}, u_{i+1,L})$ and $(u_{i,R}, u_{i+1,R})$ for every $0 \leq i < p+1$. We first fix $\eta'(v) = \eta(v)$ for every vertex $v \in V(H) - (U_L \cup U_R)$. We then embed the vertex $u_{i,L}$ (respectively, $u_{i,R}$) in the point $\eta'(u_{i,L}) = \eta(u_i) + \vec{e}_i$ (resp., $\eta'(u_{i,R}) = \eta(u_i) - \vec{e}_i$), where \vec{e}_i is a vector of sufficiently small magnitude that points to the left with respect to the direction of P at u_i . More formally, we take an infinitesimal ϵ (significantly smaller than the Euclidean distance between any pair of disjoint graph objects under η) and fix $|\vec{e}_i| = \epsilon$ for every $0 \leq i \leq p+1$. The planar embedding η' is then determined by fixing $\theta(\vec{e}_0) = \theta(\eta(u_1) - \eta(u_0)) + \pi/2$; $\theta(\vec{e}_{p+1}) = \theta(\eta(u_{p+1}) - \eta(u_p)) + \pi/2$; and $\theta(\vec{e}_i) = \theta(\eta(u_{i+1}) - \eta(u_{i-1})) + \pi/2$ for every $1 \leq i \leq p$ (see Figure 3).

Note that the new edges added to G'' in the construction of H form an undirected simple cycle C in H whose vertices are exactly those in $U_L \cup U_R$. It is straightforward to verify that the shape enclosed by C under η' does not contain any vertex other than the vertices of C . Therefore C is a face under η' and the argument holds. \square

The proof of Lemma 5.1 does not provide a fast construction of the planar embedding of H . However, from an algorithmic perspective, a planar representation¹ of H , under which H is indeed a

¹ A planar representation of a graph is a data structure that depicts the circular ordering of the neighbors of each

faced graph, can be obtained from a planar representation of G in linear time. This establishes the following corollary.

Corollary 5.2. *The path avoiding distance oracle can be implemented by an $O(n \log n)$ preprocessing stage.*

Acknowledgments. The authors would like to thank Philip Klein for his assistance.

vertex with respect to some fixed planar embedding. A corresponding planar embedding can be constructed in linear time.

References

- [1] T. H. Cormen, C. E. Leiserson and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.
- [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] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
- [4] D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998.
- [5] I. Fáry. On straight-line representation of planar graphs. *Acta Sci. Math. (Szeged)*, 11:229–233, 1948.
- [6] H. de Fraysseix, J. Pach and R. Pollack. Small sets supporting Fary embeddings of planar graphs. In *Proc. ACM Symposium on Theory of Computing (STOC)*, pages 426–433, 1988.
- [7] J. Fakcharoenphol and S. Rao. Planar graphs, negative weight edges, shortest paths, near linear time. In *Proc. IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 232–241, 2001.
- [8] J. Hershberger and S. Suri. Vickrey prices and shortest paths: what is an edge worth? In *Proc. of 42nd FOCS*, pages 252–259, 2001.
- [9] 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.
- [10] N. Katoh, T. Ibaraki, and H. Mine. An efficient algorithm for K shortest simple paths. *Networks*, 12(4):411–427, 1982.
- [11] D. R. Karger, D. Koller, and S. J. Phillips. Finding the hidden path: time bounds for all-pairs shortest paths. *SIAM Journal on Computing*, 22:1199–1217, 1993.
- [12] P. N. Klein. Multiple-source shortest paths in planar graphs. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 146–155, 2005.
- [13] E. L. Lawler. A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18:401–405, 1971/72.
- [14] 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.
- [15] 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.
- [16] N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35:166–196, 2001.
- [17] 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.
- [18] L. Roditty. On the k -simple shortest paths in weighted directed graphs. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2007.
- [19] M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46:362–394, 1999.
- [20] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, 2004.
- [21] J. Y. Yen. Finding the K shortest loopless paths in a network. *Management Science*, 17:712–716, 1970/71.