A Simple Algorithm for Multiple-Source Shortest Paths in Planar Digraphs

Das, Debarati; Kipouridis, Evangelos; Gutenberg, Maximilian Probst; Wulff-Nilsen, Christian

Published in:
Proceedings, Symposium on Simplicity in Algorithms (SOSA)

DOI:
10.1137/1.9781611977066.1

Publication date:
2022

Document version
Peer reviewed version

Document license:
CC BY

Citation for published version (APA):
A Simple Algorithm for Multiple-Source Shortest Paths in Planar Digraphs

Debarati Das∗† Evangelos Kipouridis∗†‡ Maximilian Probst Gutenberg§ Christian Wulff-Nilsen∗¶

December 20, 2021

Abstract

Given an $n$-vertex planar embedded digraph $G$ with non-negative edge weights and a face $f$ of $G$, Klein presented a data structure with $O(n \log n)$ space and preprocessing time which can answer any query $(u, v)$ for the shortest path distance in $G$ from $u$ to $v$ or from $v$ to $u$ in $O(\log n)$ time, provided $u$ is on $f$. This data structure is a key tool in a number of state-of-the-art algorithms and data structures for planar graphs.

Klein’s data structure relies on dynamic top trees and the persistence technique as well as a highly non-trivial interaction between primal shortest path trees and their duals. The construction of our data structure follows a completely different and in our opinion significantly simpler divide-and-conquer approach that solely relies on Single-Source Shortest Path computations and primal graph contractions. Our space and preprocessing time bound is $O(n \log |f|)$ and query time is $O(\log |f|)$ which is an improvement over Klein’s data structure when $f$ has small size.

∗Department of Computer Science, University of Copenhagen, Basic Algorithms Research Copenhagen (BARC). Emails: debaratix710@gmail.com, {kipouridis,koolooz}@di.ku.dk.
†Debarati Das and Evangelos Kipouridis are supported by VILLUM Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), Denmark.
‡Evangelos Kipouridis has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 801199.
§ETH Zurich, Email: maximilian.probst@inf.ethz.ch
¶Christian Wulff-Nilsen is supported by the Starting Grant 7027-00050B from the Independent Research Fund Denmark under the Sapere Aude research career programme.
1 Introduction

In the Planar Multiple-Source Shortest Paths (MSSP) problem, an embedded digraph $G$ is given along with a distinguished face $f$, and the goal is to compute a data structure answering distance queries between any vertex pair $(u, v)$ where either $u$ or $v$ belong to $f$. Data structures for this problem are measured by the preprocessing time required to construct the data structure, the space required to store the data structure, and the query time required to answer a query.

Applications. MSSP data structures are a crucial building block for the All-Pairs Shortest Paths (APSP) problem in planar graphs where the data structure needs to be able to return the (approximate or exact) distance between any two vertices in the graph. Such data structures are often called distance oracles if the query time is subpolynomial in $n$. Broadly, there are three different APSP data structure problems that currently rely on MSSP algorithms:

- Exact Distance Oracles: In a recent series of breakthroughs, [CDW17; P G+18; Cha+19; YS21] showed that it is possible to obtain an APSP data structure that requires only space $n^{1+o(1)}$ and query time $n^{o(1)}$ where both [Cha+19] and the state-of-the-art result in [YS21] employs MSSP as a building block in their construction.

- Approximate Distance Oracles: Thorup [Tho04] presented a data structure that returns $(1 + \epsilon)$-approximate distance estimates using preprocessing time and space $\tilde{O}(n\epsilon^{-1})$ and query time $O(\log \log n + \epsilon^{-1})$. Since, his construction time was sped-up by polylogarithmic factors via improvements to the state-of-the-art MSSP data structure [Kle05].

- Exact (Dynamic) APSP: A classic algorithm by Fakcharoenphol and Rao [FR06] gives a data structure that uses $\tilde{O}(n)$ space and preprocessing time, and takes query time $\tilde{O}(\sqrt{n})$ to answer queries exactly. A variant of this algorithm further gives a data structure that processes edge weight changes to the graph $G$ in time $\tilde{O}(n^{2/3})$ while still allowing for query time $\tilde{O}(n^{2/3})$. Again, while [FR06] did not directly employ an MSSP data structure, Klein [Kle05] showed that incorporating MSSP leads to a speed-up in the logarithmic factors.

We point out that various improvements were made during the last years over these seminal results when considering different trade-offs [FMW20; LP21] or by improving logarithmic or doubly-logarithmic factors [MW10; GK18; MNW18], however, the fundamental sub-problem of MSSP is present in almost all of these articles. We emphasize that beyond the run-time improvements achieved by MSSP data structure, an additional benefit is a more modular and re-usable design that makes it simpler to understand and implement APSP algorithms.

Another key application of MSSP is the computation of a dense distance graph. An often employed strategy for a planar graph problem is to decompose the embedded planar input graph $G$ into smaller graphs using vertex separators in order to either obtain a recursive decomposition or a flat so called $r$-division of $G$. For each subgraph $R$ obtained, let $\partial R$ denote its set of boundary vertices (vertices incident to edges not in $R$). The dense distance graph of $R$ is the complete graph on $\partial R$ where each edge $(u, v)$ is assigned a weight equal to the shortest path distance from $u$ to $v$ in $R$. Since the recursive or flat decomposition can be done in such a way that $\partial R$ is on a constant number of faces of $R$, MSSP can then be applied to each such face to efficiently obtain the

---

We use $\tilde{O}(\cdot)$-notation to suppress logarithmic factors in $n$. To state the bounds in a clean fashion, we assume that the ratio of smallest to largest edge weight in the graph is polynomially bounded in $n$. 

1
dense distance graph of $R$. There are numerous applications of dense distance graphs, not only for shortest path problems but also for problems related to cuts and flows [Bor+17; BSW15; Ita+11].

**Previous Work.** The MSSP problem was first considered implicitly by Fakcharoenphol and Rao [FR06] who gave a data structure that requires $\tilde{O}(n)$ preprocessing time and space and query time $\tilde{O}(\sqrt{n})$. Since, the problem has been systematically studied by Klein [Kle05] who obtained a data structure with preprocessing time and space $O(n \log n)$ and query time $O(\log n)$. Klein’s seminal result was later proven to be tight in $n$ for preprocessing time and space [EK13]. Klein and Eisenstat [EK13] also demonstrated that one can remove all logarithmic factors in the special case of undirected, unit-weighted graphs. Finally, [Kle05] was generalized by Cabello, Chambers and Erickson [CCE13] to surface-embedded graphs of genus $g$, with preprocessing time/space $\tilde{O}(gn \log n)$ and query time $O(\log n)$.

**The Seminal Result by Klein [Kle05].** On a high-level, the result by Klein [Kle05] is obtained by the observation that moving along a face $f$ with vertices $v_1, v_2, \ldots, v_k$, from vertex $v_i$ to $v_{i+1}$, the difference between the shortest path trees $T_{v_i}$ and $T_{v_{i+1}}$ consists on average of $O(n/k)$ edges. [Kle05] therefore suggests to dynamically maintain a tree $T$, initially equal to the shortest path tree $T_{v_1}$ of $v_1$, and then to make the necessary changes to $T$ to obtain the shortest path tree $T_{v_2}$ of $v_2$, and so on for $v_3, \ldots, v_k$. Overall, this requires only $O(n)$ changes to the tree $T$ over the entire course of the algorithm while passing through all shortest path trees.

To implement changes to $T$ efficiently, Klein uses a dynamic top tree data structure to represent $T$, and uses duality of planar graphs in the concrete form of an interdigitizing tree/ tree co-tree decomposition, with the dual tree also maintained dynamically as a top tree. Finally, he uses an advanced persistence technique [Dri+89] to allow access to the shortest path trees $T_{v_i}$ for any $i$ efficiently.

Even though formalizing each of these components requires great care, the algorithm by Klein is commonly taught in courses and books on algorithms for planar graphs (see for example [KM14; Dem+11; Eri20]), but with dynamic trees and persistence abstracted to black box components.

**Our Contribution.** We give a new approach for the MSSP problem that we believe to be significantly simpler and that matches (and even slightly improves) the time and space bounds of [Kle05]. Our algorithm only uses the primal graph, and consist of an elegant interweaving of Single-Source Shortest Paths (SSSP) computations and contractions in the primal graph.

Our contribution achieved via two variations of our MSSP algorithm comprises of:

- **A Simpler, More Accessible Data Structure:** We give a MSSP data structure with preprocessing time/space $O(n \log |f|)$ and query time $O(\log |f|)$ which slightly improves the state-of-the-art result by Klein [Kle05] for $|f|$ subpolynomial and otherwise recovers his bounds. Our result is achieved by implementing SSSP computations via the linear-time algorithm for planar digraphs by Henzinger et al. [Hen+97]. Abstracting the algorithm [Hen+97] in black-box fashion, our data structure is significantly simpler than [Kle05] and we believe that it can be taught at an advanced undergraduate level.

---

2[EK13] also shows how to use this data structure to obtain a linear-time algorithm for the Max Flow problem in planar, unit-weighted digraphs
Further, by replacing the black-box from [Hen+97] with a standard implementation of Dijkstra’s algorithm, our algorithm can easily be taught without any black-box abstractions, at the expense of only an $O(\log n)$-factor to the preprocessing time.

- **A More Practical Algorithm:** We also believe that our algorithm using Dijkstra’s algorithm for SSSP computations is easier to implement and performs significantly better in practice than the algorithm by Klein [Kle05]. We expect this to be the case since dynamic trees and persistence techniques are complicated to implement and incur large constant factors, even in their heavily optimized versions (see [TW10]). In contrast, it is well-established that Dijkstra’s algorithm performs extremely well on real-world graphs, and contractions can be implemented straight-forwardly.

In fact, one of the currently most successful experimental approaches to compute shortest-paths in road networks is already based on a framework of clever contraction hierarchies and fast SSSP computations (see for example [Gei+08]), and it is perceivable that our algorithm can be implemented rather easily by adapting the components of this framework.

We point out that our result can be shown to be tight in $|f|$ and $n$ by straight-forwardly extending the lower bound in [EK13]. We can report paths in the number of edges plus $O(\log |f|)$ time.

## 2 Preliminaries

Given a graph $H$, we use $V(H)$ to refer to the vertices of $H$, and $E(H)$ to refer to its edges. We denote by $w_H(e)$ or $w_H(u, v)$ the weight of edge $e = (u, v)$ in $H$, by $d_H(u, v)$ the shortest distance from $u$ to $v$ in $H$ and by $P_H(u, v)$ a shortest path from $u$ to $v$. By SSSP tree from a vertex $u \in V(H)$, we refer to the shortest path tree from $u$ in $H$ obtained by taking the union of all shortest paths starting in $u$ (where we assume shortest paths satisfy the subpath property). We use $T(r)$ to denote a tree rooted at a vertex $r$. For a vertex $v \neq r$ of $T$, we let $\pi_T(v)$ denote the parent of $v$ in $T$.

**Induced Graph/Contractions.** For a vertex set $X \subseteq V(H)$, we let $H[X]$ denote the subgraph of $H$ induced by $X$. We sometimes abuse notation slightly and identify an edge set $E'$ with the graph having edges $E'$ and the vertex set consisting of endpoints of edges from $E'$. For any edge set $E' \subseteq E(H)$, we let $H/E'$ denote the graph obtained from $H$ by contracting edges in $E'$ where we remove self-loops and retain only the cheapest edge between each vertex pair (breaking ties arbitrarily). If $E$ only contains a single edge $(u, v)$, we slightly abuse notation and write $H/(u, v)$ instead of $H/\{(u, v)\}$. When we contract components of vertices $x_1, x_2, \ldots, x_k$ into a super-vertex, we will identify the component with some vertex $x_i$. We use the convention that when we refer to some vertex $x_j$ from the original graph in the context of the contracted graph, then $x_j$ refers to the identified vertex $x_i$.

**The Input.** We let $G = (V, E)$ refer to the input graph in this article, and assume that $G$ is a planar embedded graph where we assume that the embedding is given by a standard rotation system, meaning that neighbors of a vertex are ordered clockwise around it. We assume that $G$ is strongly-connected, has unique shortest paths and that $f$ given as input is embedded as the infinite face. All assumptions are without loss of generality (see in particular the standard perturbation technique for unique shortest paths [EFL18]).
3 The Data Structure

Preprocessing of $G$. To ease the description of our data structure, we preprocess inputs $G$ and $f$ as follows. We let $b_0, b_1, \ldots, b_{|f| - 1}$ be the vertices on $f$ ordered by their first appearance in the walk along $f$ in clockwise order starting in an arbitrary vertex $b_0$, such that the rest of $G$ is on the right when moving on the walk. For each $0 \leq i < |f|$, we add a vertex $r_i$, and a zero-weight edge $(r_i, b'_i)$ embedded in the region enclosed by $f$ that does not contain the rest of $G$. Finally, we add edges $(r_i, r_{i+1 \mod |f|})$ in a simple cycle $C$ again with infinity weights. We let $f_\infty$ be the face associated with the region enclosed by the cycle $C$ that does not contain $G$ and denote by $V_\infty$ the vertex set of $f_\infty$. Note that by construction $f_\infty$ is the new infinite face. We denote the new graph by $G_\infty$. We now show how to construct our MSSP data structure on $G_\infty$ assuming that all queries are on pairs $(u, v) \in V_\infty \times V(G)$. It is not hard to see that any query $(b_i, u)$ to $G$ can then be mapped to a query $(r_i, u)$ on $G_\infty$.

Algorithm 1: The procedure MSSP is given an interval $I = [i_1, i_2]$ and a graph $H_I$ obtained from contracting edges in $G$ in the graph $G_\infty$ (thus $f_\infty$ is preserved in $H_I$). $H_I$ contains roots $r_j$ for $j \in I$. The initial call is to $([0, |f_\infty| - 1], G_\infty)$.

\begin{algorithm}
Procedure MSSP($I = [i_1, i_2]$, $H_I$)
1\begin{algorithmic}
2\STATE $i \leftarrow \left\lceil \frac{i_1 + i_2}{2} \right\rceil$
3\FOREACH $k \in \{i_1, i_2, i\}$
4\STATE Compute and store SSSP tree $T_{I,k}$ in $H_I[(V(H_I) - V_\infty) \cup \{r_k\}]$ from $r_k$
5\STATE if $i_2 - i_1 \leq 1$ then return
6\STATE $J = [j_1, j_2] \in \{[i_1, i], [i, i_2]\}$ do
7\STATE $H_J \leftarrow H_I$
8\STATE $E_{\text{shared}} \leftarrow E(T_{I,j_1}) \cap E(T_{I,j_2})$
9\STATE Let $T$ be the collection of maximal vertex-disjoint trees $T(s)$ rooted at $s$ in $(V(H_I) - V_\infty, E_{\text{shared}})$ such that $\pi_{T_{I,j_1}}(s) \neq \pi_{T_{I,j_2}}(s)$, and for each child $v$ of $s$ in $T(s)$ the edges $(s, v), (s, \pi_{T_{I,j_1}}(s)), (s, \pi_{T_{I,j_2}}(s))$ are clockwise around $s$
10\STATE foreach $T(s) \in T$ do $H_J \leftarrow \text{CONTRACT}(H_J, T(s), i)$
11\STATE MSSP($J, H_J$)
12\ENDFOREACH
13\ENDProcedure

Procedure CONTRACT($H_J, T(s), i$)
14\BEGIN
15\FOREACH vertex $u \in T(s)$ do $(s_1(u), \delta_1(u)) \leftarrow (s, d_{T(s)}(s, u))$ // Global variables
16\FOREACH $(u, v) \in E(H_J)$ with exactly one endpoint in $T(s)$ do
17\IF $v \notin T(s)$ then $w_{H_J}(u, v) \leftarrow w_{H_J}(u, v) + \delta_1(u)$ // $(u, v)$ outgoing from $T(s)$
18\ELSEIF $v \neq s$ then $w_{H_J}(u, v) \leftarrow \infty$ // $(u, v)$ ingoing to $T(s) - \{s\}$
19\ENDIF
20\ENDFOREACH
21\Contract $T(s)$ to a vertex in $H_J$ and identify it with $s$
22\RETURN $H_J$
23\ENDProcedure

The MSSP Data Structure. Let us now discuss the algorithm to construct the MSSP data structure which is given in Algorithm 1. The procedure MSSP($I = [i_1, i_2], H_I$) starts by partitioning the interval $I$ into two roughly equally sized subintervals $[i_1, \bar{i}]$ and $[\bar{i}, i_2]$, in Lines 1-3. It then computes the shortest path trees of the boundary vertices $r_{i_1}, r_{i_2}, \bar{i}$ in the graph $H_I$ (after
removing the other boundary vertices). If \( i_2 - i_1 > 1 \), the data structure is recursively built for each of the two subintervals \( J = [j_1, j_2] \) in the loop starting in Line 5. To get the desired preprocessing time, the data structure ensures that the total size of all graphs at a given recursion level is \( O(n) \). For each subinterval \( J \), this is ensured by letting the graph \( H_J \) for the recursive call be a suitable contraction of \( H_I \). More precisely, \( H_J \) is obtained from \( H_I \) by contracting suitable edges \( e \) that are guaranteed to be in the SSSP trees of every root \( r_j \) with \( j \in J \). The construction of \( T \) and the procedure \textsc{Contract} handle the details of these contractions (illustrated in Figure 1). They also store in Line 11 the information necessary for later queries.

Figure 1: To the left, we have tree \( T(s) \). To the right we contract \( T(s) \) and identify the supervertex with \( s \). We update the weight of outgoing edges and remove the edges ingoing in \( T(s) - s \).

A small but important implementation detail, we point out, is that while the pseudocode specifies that Algorithm 1 initializes each graph \( H_J \) as a copy of \( H_I \) in Line 6, the data structure technically only copies the subset \( \{ r_{j_1}, r_{j_1+1}, \ldots, r_{j_2} \} \) of the vertices on \( f_\infty \). It is clear from the pseudocode that the omitted roots are not part of any shortest path tree in any recursive calls involving sub-intervals of \( J \) so omitting them will not affect the behaviour of \textsc{MSSP}. Including the entire face \( f_\infty \) in all recursive calls, however, eases the presentation of proofs.

Query. The query procedure is straight-forward and given by Algorithm 2.

\begin{algorithm}
\caption{The procedure to query \( d_G(b_j, u) \). Initial call is \textsc{Query}(\( u, j, [0, |f_\infty| - 1] \)).}
\begin{algorithmic}[1]
\Procedure{\textsc{Query}}{\( u, j, I = [i_1, i_2] \)}
\If {\( j = i_1 \) or \( j = i_2 \)} \Return \( d_{T_{i,j}}(r_j, u) \)
\EndIf
\State \( i \leftarrow \frac{i_1 + i_2}{2} \)
\If {\( j \leq i \)} \Return \textsc{Query}(\( s_i(u), j, [i_1, i] \)) + \( \delta_i(u) \)
\EndIf
\Else  \Return \textsc{Query}(\( s_i(u), j, [i, i_2] \)) + \( \delta_i(u) \)
\EndProcedure
\end{algorithmic}
\end{algorithm}

4 Analysis

We now prove the following theorem which summarizes our main result.

**Theorem 4.1.** Let \( G \) be an \( n \)-vertex planar embedded graph and \( f \) be the infinite face on \( G \). Then we can build a data structure answering the distance \( d_G(b_j, u) \) between any vertex \( b_j \in f \) and any other vertex \( u \) in \( O(\log |f|) \) time, using procedure \textsc{Query}(\( u, j, [0, |f| - 1] \))). Preprocessing requires \( O(n \log |f|) \) time and space, using procedure \textsc{MSSP}(\( [0, |f_\infty| - 1], G_\infty \)).
Figure 2: The shortest path trees from \( r_{j_1} \) and \( r_{j_2} \) share the dashed edges denoting the tree \( T(s) \) containing \( u \). The shortest \( r_{j_1}\text{-to-}s \), \( r_{j_2}\text{-to-}s \) paths and the subpath from \( r_{j_2} \) to \( r_{j_1} \) in clockwise order along \( f_\infty \) define \( C \) (fat line). As \( v, v_1, v_2 \) are in clockwise order around \( s \), \( T(s) - s \) is in \( R \). For \( j \in [j_1, j_2] \), any \( r_j\text{-to-}u \) path intersects either the shortest \( r_{j_1}\text{-to-}s \) or the shortest \( r_{j_2}\text{-to-}s \) path.

**Correctness.** Let us first prove correctness of the data structure. We start with the observation that no edge incident to the infinite face is ever contracted.

**Claim 4.2.** For any graph \( H_I \), no edge incident to its infinite face is contracted by MSSP\((I, H_I)\).

**Proof.** Observe that any edge that is contracted in some \( H_I \) has to be in at least two of the graphs \( H_I[(V(H_I) - V_\infty) \cup \{r_k\}] \) for \( k \in \{i_1, j, i_2\} \) and \( i_1 < j < i_2 \). But since the only edge incident to \( V_\infty \) in each such graph is the edge \((r_k, b_k)\), no edge on \( f_\infty \) is contracted.

Next, we prove a lemma and its corollary that roughly show that the contractions made in Procedure MSSP do not destroy shortest paths from roots on sub-intervals \( J \) of \( f_\infty \). The reader is referred to Figure 2 for intuition. We first need the following simple claim.

**Claim 4.3.** For any invocation of MSSP\((I, H_I)\), any \( i \in I \), and any \( u \in V(H_I) - V_\infty \), some \( r_i\text{-to-}u \) path exists in \( H_I[(V(H_I) - V_\infty) \cup \{r_i\}] \).

**Proof.** Edge \((r_i, b_i)\) exists in \( G_\infty \) and is not contracted, the subgraph \( G \) of \( G_\infty \) is strongly-connected by assumption, and \( H_I \) is obtained from contractions in \( G_\infty \) while preserving \( f_\infty \) by Claim 4.2.

**Lemma 4.4.** Consider any invocation of MSSP\((I, H_I)\) where \( H_I \) has unique shortest paths for paths starting in \( r_i \) for each \( i \in I \). Then for each vertex \( u \in T(s) \in T \) in Line 8 and for each \( j \in J \), we have \( P_{H_i}[s, u] \subseteq P_{H_i}[r_j, u] \).

**Proof.** The proof is trivial for \( u = s \), thus we assume \( u \neq s \). Now consider paths \( P_{T_{I,j_1}}[r_{j_1}, s] \) and \( P_{T_{I,j_2}}[r_{j_2}, s] \). Both paths exist by Claim 4.3. By uniqueness of shortest paths in \( H_i \), they share only vertex \( s \).

Next, consider the concatenation \( P[j_1, j_2] \) of \( P_{T_{I,j_1}}[r_{j_1}, s] \) and the reverse of \( P_{T_{I,j_2}}[r_{j_2}, s] \). From the above, \( P[j_1, j_2] \) is simple. Further, consider the concatenation \( C \) of \( P[j_1, j_2] \) and the path segment \( F_\infty[r_{j_2}, r_{j_1}] \) from \( r_{j_2} \) to \( r_{j_1} \) in clockwise order around \( f_\infty \). We note that \( C \) is a cycle. As \( f_\infty \) is simple so is \( F_\infty[r_{j_2}, r_{j_1}] \), and since \( f_\infty \) has no ingoing edges from \( H_i \) (by construction of \( G_\infty \)}.  

6
and by Claim 4.2, $P[j_1, j_2]$ only intersects $f_\infty$ in vertices $r_{j_1}$ and $r_{j_2}$. Thus, $C$ is a simple directed cycle.

By the Jordan curve theorem, $C$ partitions the plane into two regions. One region, denoted $R$, is the region to the right when walking along $C$. $T(s) - s$ does not intersect the simple curve $P[j_1, j_2]$ since $H_I$ is a planar embedded graph and since shortest paths are simple. It also does not intersect $F_\infty[r_{j_2}, r_{j_1}]$ as there are no ingoing edges to $f_\infty$ in $G_\infty$ (and $f_\infty$ is preserved in contractions by Claim 4.2). Since for each child $v$ of $s$ in $T(s)$, the edges $(s, v), (s, \pi_{T_I,j_1}(s)), (s, \pi_{T_I,j_2}(s))$ are clockwise around $s$, children of $s$ are contained in $R$, and hence so is $T(s) - s$.

By the choice of $F_\infty[r_{j_2}, r_{j_1}]$, $r_j$ does not belong to $R$ so $P_{H_I}[r_j, u]$ intersects $C$. Since $u \notin F_\infty[r_{j_2}, r_{j_1}]$ and there is no edge $(a, b)$ of $H_I$ with $a \notin V_\infty$ and $b \in V_\infty$, $P_{H_I}[r_j, u]$ cannot intersect $F_\infty[r_{j_2}, r_{j_1}] - \{r_{j_1}, r_{j_2}\}$ so it must intersect $C$ in $P[j_1, j_2]$ and hence intersect either $P_{T_I,j_1}[r_{j_1}, s]$ or $P_{T_I,j_2}[r_{j_2}, s]$; assume the former (the other case is symmetric). Then $P_{H_I}[r_j, u]$ intersects $P_{T_I,j_1}[r_{j_1}, s]$ in some vertex $x$. Since shortest paths are unique, the subpath of $P_{H_I}[r_j, u]$ from $x$ to $u$ must then equal $P_{T_I,j_1}[x, u]$ and since $s$ is on this subpath, the lemma follows.

**Corollary 4.5.** Let $H_J$ be one of the graphs obtained in a call to MSSP($I, H_I$) by contracting edges $E'$ in $H_I$. Then, for each $j \in J$ and each $u \in V(H_J) - V_\infty$, $P_{H_I}[r_j, u] = P_{H_I}[r_j, u]/E'$ is unique and $w_{H_J}(P_{H_I}[r_j, u]) = w_{H_I}(P_{H_I}[r_j, u])$.

**Proof.** By Lemma 4.4, when contracting $P_{H_I}[r_j, u]$ by $E'$, no edge on the path has its weight increased to $\infty$. Further, for any edge $(x, y) \in P_{H_I}[r_j, u]$, either $(x, y)$ existed already in $H_I$ in which case its weight is unchanged, or $(x, y)$ originated from an edge $(w, y)$ for $w \in T(x)$. But in the latter case, $P_{H_I}[r_j, u]$ contains $P_{H_I}[x, w]$ followed by $(w, y)$ (Lemma 4.4), whose weight is $d_{H_I}(x, w) + w_{H_I}(w, y) = d_{H_J}(x, y)$. Thus $w_{H_J}(P_{H_I}[r_j, u]) \leq w_{H_I}(P_{H_I}[r_j, u])$.

It is straightforward to see that distances in $H_J$ also have not decreased since edges affected by the contractions obtain weights corresponding to paths in $H_I$ between their endpoints or weight $\infty$. Uniqueness follows since two different shortest paths in $H_J$ from $r_j$ to a vertex $u$ would imply two different shortest paths between the same pair in $H_I$ and, by an inductive argument, also in $G_\infty$, contradicting our assumption of uniqueness.

In fact, Corollary 4.5 is all we need to prove correctness of our algorithm.

**Lemma 4.6.** The call Query($u, j, I = \{i_1, i_2\}$) outputs $d_{H_I}(r_j, u)$, for $u \in V(H_I) - V_\infty$. In particular, Query($u, j, I = [0, f_\infty - 1]$) outputs $d_{G}(b_j, u) = d_{G_\infty}(r_j, u)$, for $u \in V(G)$.

**Proof.** We prove this by induction on $i_2 - i_1$. If $j \in \{i_1, i_2\}$ then Query($u, j, [i_1, i_2]$) directly returns $d_{T_I,j}(r_j, u) = d_{H_I}(s_{i_1}(u), j, [i_1, i_2]) + \delta_i(u) = d_{H_I}(r_j, s) + d_{T(s)}(s, u)$. By Corollary 4.5, $d_{H_I}(r_j, s) = d_{H_I}(r_j, s)$. By definition of $T(s)$, $d_{T(s)}(s, u)$ is a suffix of $d_{H_I}(r_{i_1}, u)$, meaning that $d_{T(s)}(s, u) = d_{H_I}(s, u)$. Finally, by Lemma 4.4 the shortest path in $H_I$ from $r_j$ to $u$ contains $s$, therefore $d_{H_I}(r_{i_1}, j)(r_j, s) + d_{T(s)}(s, u) = d_{H_I}(r_j, s) + d_{H_I}(s, u) = d_{H_I}(r_j, u)$.
Bounding Time and Space. The following Lemma captures our key insight about Algorithm 1 that ensures that the algorithm can be implemented efficiently.

Definition 4.7. Let \( \mathcal{I}_h \) be the set of all intervals \( I \), such that \( \text{MSSP}(I, H_I) \) is executed at recursion level \( h \) after invoking \( \text{MSSP}([0, |f_\infty| - 1], G_\infty) \).

Lemma 4.8. For each edge \( e = (u, v) \in E(G) \) and recursion level \( h \), there are at most \( O(1) \) intervals \( I \in \mathcal{I}_h \) for which there exists an \( i \in I \) such that the SSSP tree \( T_{I,i} \) from \( r_i \) in \( H_I \) \(|V(H_I) - V_\infty| \cup \{r_i\}\) contains \( e \).

Proof. We use induction on \( h \). As \( \mathcal{I}_0 = \{[0, |f_\infty| - 1]\} \) the Lemma is trivial for level \( h = 0 \). For \( h \geq 0 \), we show the inductive step \( h \mapsto h + 1 \). Each interval \([i_1, i_2] = I \in \mathcal{I}_h \) satisfies exactly one of the following conditions:

- \( \forall i \in I, e \notin E(T_{I,i}) \): Consider a recursive call \( \text{MSSP}(J, H_J) \) for \( J \subseteq I \) issued in \( \text{MSSP}(I, H_I) \) where \( H_J \) was obtained by contracting edges \( E' \) in \( H_I \). By Corollary 4.5, for any \( j \in J \), \( P_{H_J}[r_j, u] = P_{H_J}[r_j, u]/E' \), but due to our condition and \( j \in I \), this path cannot contain \( e \).

- \( \forall i \in I, e \in E(T_{I,i}) \): Consider each iteration of the foreach-loop in Line 5 in \( \text{MSSP}(I, H_I) \). Let \( J \subseteq I \) be one of the at most two intervals considered during one such iteration.

Note that if \( e \in T(s) \) for some \( T(s) \in T \), then to construct \( H_J \) from \( H_I \) the edges in \( T(s) \) are contracted in line 15, and therefore \( e \) cannot be contained in \( H_J \).

We therefore assume that \( e \notin T(s) \) for any \( T(s) \). We claim that there is at most one interval \([j_1, j_2] \in \mathcal{I}_{h+1} \) that contains \( e \) under these conditions. The proof of this claim is illustrated in Figure 3.

![Figure 3: The dashed lines represent shortest paths \( P_{G_\infty}[r_i, u] \) from a vertex \( r_i \) in \( V_\infty \) to \( u \). In the second level of recursion we have intervals \( I_0 = [0, 2], I_1 = [2, 4], I_2 = [4, 6], I_3 = [6, 8] \). \( C_{0,2,u} \) is a simple cycle defined by \( P_{G_\infty}[0, u], P_{G_\infty}[2, u] \) and part of \( f_\infty \). Similarly for \( C_{2,4,u}, C_{4,6,u}, C_{6,8,u} \). Focusing on interval \( I_1 \), the three edges \((u, v), (u, \pi_{P_{G_\infty}[r_2,u]}(u)), (u, \pi_{P_{G_\infty}[r_4,u]}(u)) \) are in counterclockwise order, meaning that \((u, v)\) belongs to the region of the plane enclosed by \( C_{2,4,u} \). None of the other cycles can have this property.](image)

For the sake of contradiction, assume that there are at least two multiple such intervals of this type and let us analyze what must hold for each of them. Let Assume at least one such
interval \([j_1, j_2]\) be one such interval exists (otherwise, we are done) and let us analyze what must hold for this interval. By the main condition, the paths \(P_{H_1}[r_{j_1}, v]\) and \(P_{H_1}[r_{j_2}, v]\) share the edge \((u, v)\). We let \((x, y)\) be the first edge these two paths share. It is clear that paths \(P_{G_\infty}[r_{j_1}, v]\) and \(P_{G_\infty}[r_{j_2}, v]\) also share \((u, v)\) (by Corollary 4.5). We let \((x', y')\) be the first edge that these two paths share. Finally, we let \(I_0 = [0, |f_\infty| - 1]\).

We claim that \((x', y'), (x', \pi_{T_{j_0,j_1}}(x'))\), \((x', \pi_{T_{j_0,j_2}}(x'))\) are counter-clockwise around \(x'\) in \(G_\infty\) (where the parents of \(x'\) exist by Claim 4.2 and construction of \(G_\infty\)). Consider otherwise, then \((x, y), (x, \pi_{T_{j_0,j_1}}(x)), (x, \pi_{T_{j_0,j_2}}(x))\) are clockwise around \(x\) in \(H_1\), since the edges on the path segment in \(G_\infty\) from \(x'\) to \(x\) must be contracted in \(H_1\) (by minimality of \((x, y)\) and Corollary 4.5), and by the embedding-preserving properties of contractions. By the definition of \(T\) in Line 8, it follows that \((x, y)\) is contained in tree \(T(x) \in T\). Since further \(E_{\text{shared}} \supseteq P_{H_1}[r_{j_1}, v] \cap P_{H_1}[r_{j_2}, v]\), and the path segment from \(x\) to \(v\) is shared, the maximality of trees in \(T\) implies that these edges are in \(T(x)\), and in particular \(e \in T(x)\) contradicting our sub-condition.

Now, let us fix two distinct intervals \([j_1, j_2]\) and \([j'_1, j'_2]\) that we assume exist. They both satisfy the counter-clockwise property shown above as defined above. Let \(C\) be the directed cycle formed by the concatenation of \(P_{G_\infty}[r_{j_1}, x']\), the reverse path of \(P_{G_\infty}[r_{j_2}, x']\) and the path from \(r_{j_2}\) to \(r_{j_1}\) in counter-clockwise order along \(f_\infty\) (see examples of such cycles in Figure 3). Then it is not hard to see that \(C\) is simple (using the same argument as in the proof of Lemma 4.4) and. Then \(v\) is contained in the region \(L\) that is to the left of the cycle (when walking in the direction of the cycle); this follows since \((x', y'), (x', \pi_{T_{j_0,j_1}}(x')), (x', \pi_{T_{j_0,j_2}}(x'))\) are counter-clockwise around \(x'\).

Define analogously for \([j'_1, j'_2]\), the first shared edge \((x'', y'')\) in \(G_\infty\), the cycle \(C'\) and the region \(L'\) left of \(C'\). Then \(L\) and \(L'\) are disjoint since otherwise, either \(P_{G_\infty}[r_{j'_1}, x'']\) or \(P_{G_\infty}[r_{j'_2}, x'']\) cross \(C\), contradicting the sub-path property of shortest paths. Since the subpath from \(x''\) to \(v\) shared by the two paths has to enter \(L\), we can finally conclude that \((x'', y''), (x'', \pi_{T_{j'_0,j'_1}}(x''))\), \((x'', \pi_{T_{j'_0,j'_2}}(x''))\) are clockwise around \(x''\). But this is a contradiction since we argued above that the order should be counter-clockwise, which gives the desired contradiction.

- \(\exists i, i' \in I, e \in E(T_{I, i})\) and \(e \notin E(T_{I, i'})\): We claim that at most two intervals \(I \in I_h\) satisfy this condition which can then spawn at most 4 intervals in \(I_{h+1}\).

We first prove that for any two distinct \(r_{i_1}, r_{i_2} \in V_\infty, i_1 \leq i_2\) where \(e \in P_{G_\infty}[r_{i_1}, v] \cap P_{G_\infty}[r_{i_2}, v]\), we have that either all \(i \in [i_1, i_2]\) or all \(i \in [i_2, |f_\infty| - 1]\) \(\cup [0, i_1]\) have \(e \in P_{G_\infty}[r_{i}, v]\).

To this end, let \((x, y)\) be the first shared edge on \(P_{G_\infty}[r_{i_1}, v]\) and \(P_{G_\infty}[r_{i_2}, v]\), and again \(I_0 = [0, |f_\infty| - 1]\). Let \(C\) be the directed simple cycle consisting of \(P_{G_\infty}[r_{i_1}, x]\), the reverse of \(P_{G_\infty}[r_{i_2}, x]\) and the clockwise walk from \(r_{i_2}\) to \(r_{i_1}\) along \(f_\infty\). Note that \(C\) (viewed as a simple closed curve) separates \(y\) from every root \(r_i\) with \(i \in [i_1, i_2]\). Thus, in the case where the edges \((x, y), (x, \pi_{T_{i_0,i_1}}(x)), (x, \pi_{T_{i_0,i_2}}(x))\) are clockwise around \(x\), each \(P_{G_\infty}[r_{i}, v]\) contains \(x\) and uses the same subpath as \(P_{G_\infty}[r_{i_1}, v]\) from \(x\) to \(v\); in particular, it contains \((u, v)\).

The case where \((x, y), (x, \pi_{T_{i_0,i_1}}(x)), (x, \pi_{T_{i_0,i_2}}(x))\) are counter-clockwise around \(x\) is analogous using instead the simple cycle consisting of \(P_{G_\infty}[r_{i_1}, x]\), the reverse of \(P_{G_\infty}[r_{i_2}, x]\) and the counter-clockwise walk from \(r_{i_2}\) to \(r_{i_1}\) along \(f_\infty\).
The above implies that the roots of $f_\infty$ whose shortest path trees in $G_\infty$ contain $e$ are consecutive in the cyclic ordering along $f_\infty$. This immediately implies that at most two intervals $I \in \mathcal{I}_h$ satisfy the condition above.

\begin{corollary}
\label{corollary4.9}
The procedure $\text{MSSP}([0, |f_\infty| - 1], G_\infty)$ uses $O(n \log |f|)$ time and space. Procedure $\text{Query}(u, j, [0, |f_\infty| - 1])$ has $O(\log |f|)$ running time.
\end{corollary}

\begin{proof}
For each recursion level $h$, Lemma \ref{lemma4.8} implies that each edge $e \in E(G)$ appears in at most $O(1)$ computed SSSP trees. Further, each SSSP tree uses exactly one edge from $E(G_\infty) - E(G)$, namely the edge from the root $r_i$ to $b_i$, since we remove vertices on $f_\infty$ (except for $r_i$) during the computation. Thus $O(\sum_{I \in \mathcal{I}_h}(|H_I| - |V_\infty| + 1)) = O(n)$. 

Each SSSP computation from $r$ in $H_I[(V(H_I) - V_\infty) \cup \{r\}]$ can be implemented in time $O(|H_I| - |V_\infty| + 1)$ by [Hen+97]. For each $H_I$, we also spend time $O(|H_I| - |V_\infty| + |I|)$ on constructing the graphs $H_J$ (or more precisely $H_I[(V(H_I) - V_\infty) \cup \{r_{j_1}, \ldots, r_{j_2}\}]$) since the set $E_{\text{shared}} \subseteq E(H_I)$ and associated trees $T(s)$ can be found in $O(1)$ time per edge. Each edge is outgoing from at most one and ingoing to at most one contracted tree and thus possibly changing its weight requires only $O(1)$ time. The number of edges in $H_I[(V(H_I) - V_\infty) \cup \{r_{j_1}, \ldots, r_{j_2}\}]$ not in $E(G)$ is $O(|I|) = O(|I|)$. Finally, we have $O(\sum_{I \in \mathcal{I}_h} |I|) = O(|V_\infty|)$. Since each subscript of any element $r_i$ in $V_\infty$ is in at most two intervals of $\mathcal{I}_h$. It follows that the entire time spent on recursion level $h$ is $O(n)$. 

Since there are at most $O(\log |f_\infty|)$ levels, the first part of the Corollary follows. The second part follows since each recursive step in $\text{QUERY}$ takes constant time.
\end{proof}
References


