Discrete Applied Mathematics 346 (2024) 183–191

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Non-crossing shortest paths lengths in planar graphs in linear time☆

Lorenzo Balzotti *, Paolo G. Franciosa

Dipartimento di Scienze Statistiche, Sapienza Università di Roma, p.le Aldo Moro 5, 00185 Roma, Italy

ARTICLE INFO

Article history:
Received 22 March 2023
Received in revised form 30 November 2023
Accepted 10 December 2023
Available online 23 December 2023

Keywords:
Shortest paths
Undirected planar graphs
Non-crossing paths

ABSTRACT

Given a plane graph it is known how to compute the union of non-crossing shortest paths. These algorithms do not allow neither to list each single shortest path nor to compute length of shortest paths. Given the union of non-crossing shortest paths, we introduce the concept of *shortcuts* that allows us to establish whether a path is a shortest path by checking local properties on faces of the graph. By using shortcuts we can compute the length of each shortest path, given their union, in total linear time, and we can list each shortest path p in $O(\max\{\ell, \ell \log\log \frac{k}{\ell}\})$, where ℓ is the number of edges in p and k the number of shortest paths.

1. Introduction

The problem of finding non-crossing shortest paths in a plane graph (i.e., a planar graph with a fixed planar embedding) has its primary applications in VLSI layout [7,15,16], where two paths are *non-crossing* if they do not cross each other in the chosen embedding. Thanks to Reif [20], it also appears as a basic step in the computation of maximum flow in a planar network and related problems [1,5]. The *non-crossing shortest path* (NCSP) problem can be formalized as follows: given an undirected plane graph G with non-negative edge lengths and k terminal pairs that lie on a specified face boundary, w.l.o.g, the external face boundary, find k non-crossing shortest paths in G, each connecting a terminal pair. It is assumed that terminal pairs appear in the infinite face so that non-crossing paths exist; this property can be easily verified in linear time.

We deal with a problem strictly linked to the NCSP problem. Our input is an undirected plane graph U composed by the union of k non-crossing shortest paths in a plane graph G whose extremal vertices lie on the same face of G. Thus U arises from the union of paths $p_1, \ldots, p_k$, where, for each $i \in [k]$, p_i is a shortest path from x_i to y_i in G, x_i and y_i are on the infinite face f^∞ of G, p_i and p_j are non-crossing for every $i, j \in [k]$. We stress that we know U but we ignore the p_i's; indeed, algorithms solving the NCSP problem in [3,4,21] compute the union of the non-crossing shortest paths without listing every single path.

We show how to compute the lengths of the p_i's shortest paths in linear time, i.e., in time proportional to the number of edges/vertices in U. In this way we prove that if there exists an algorithm solving the NCSP problem, then we can compute the lengths of shortest non-crossing paths in the same time complexity. We also explain an efficient way to list each path.

Our problem was already discussed in the geometrical case [17,19] under Euclidean distances, but not in a weighted plane graph which have a more general structure.

☆ A preliminary version of this paper appeared in [6].

* Corresponding author.
E-mail addresses: lorenzo.balzotti@uniroma1.it (L. Balzotti), paolo.franciosa@uniroma1.it (P.G. Franciosa).

https://doi.org/10.1016/j.dam.2023.12.011

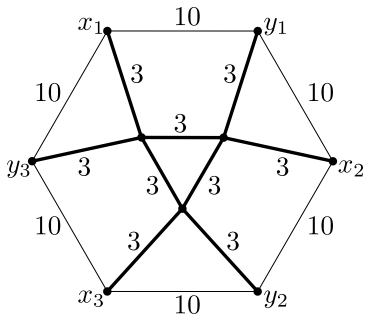

Fig. 1. In this example the union of shortest paths from x_i to y_i, for $i = 1, 2, 3$, contains a cycle (the union is highlighted with bold edges).

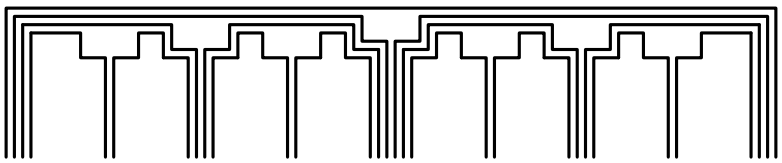

Fig. 2. Union of 15 non-crossing shortest paths that cannot be decomposed into two forests so that each path is contained in at least one forest (parallel adjacent segments represent overlapping paths).

State of the art Takahashi et al. [21] proposed an algorithm able to compute k non-crossing shortest paths that requires $O(n \log n)$ time, where n is the size of G. In the same article it is also analysed the case where the terminal pairs lie on two different face boundaries, and this case is reduced to the previous one within the same computational complexity. The complexity of their solution can be reduced to $O(n \log k)$ by plugging in the linear time algorithm by Henzinger et al. [12] for computing a shortest path tree in a planar graph. In the unweighted case, by using the result of Eisenstat and Klein [9], Balzotti and Franciosa showed that k non-crossing shortest paths can be found in $O(n)$ time [3].

Our problem is a special case of computing distances between vertices lying on the same face of a plane undirected graph. This problem can be solved in $O(n \log n)$ by Klein's algorithm [14], that has been recently improved [8] to $O(n \log |f|)$, where f is the face of G where terminal pairs lie.

The algorithm proposed in [21] first computes the union of the k shortest paths, which is claimed to be a forest. The second step relies on the data structure due to Gabow and Tarjan [11] for efficiently solving least common ancestor (LCA) queries in a forest, in order to obtain distances between the terminal pairs in $O(n)$ time.

Actually, the union of the k shortest paths may in general contain cycles. An instance is shown in Fig. 1, in which the unique set of three shortest paths forms a cycle, hence the distances between terminal pairs cannot always be computed by solving LCA queries in a forest. This limitation was noted first in [10,19].

Ericksonn and Nayyeri [10] generalized the work in [21] to the case in which the k terminal pairs lie on h face boundaries. They proved that k non-crossing paths, if they exist, can be found in $2^{O(h^2)} n \log k$ time. The authors also stated that the union of non-crossing shortest paths can always be covered with two (possibly non edge-disjoint) forests so that each path is contained in at least one forest. They do not describe how to obtain such a decomposition. This is in contrast with the example in Fig. 2, where we report the union of 15 non-crossing shortest paths that cannot be covered with two forests so that each path is contained in at least one forest (it can be easily proved by enumeration). Recently, Balzotti [2] proved that the union of non-crossing shortest paths can always be covered with four (possibly not edge-disjoint) forests so that each path is contained in at least one forest, and he also showed that four forests are necessary for some instances. We stress that the theoretical result in [2] does not provide a linear time algorithm for computing the covering forests, thus it does not allow to compute path lengths in linear time exploiting lower common ancestors computation as in [11].

Our contribution In this article we exploit the novel concept of *shortcuts*, that are portions of the boundary of a face that allow us to modify a path without increasing (and possibly decreasing) its length. We show that it is possible to establish whether a path is a shortest path by looking at the presence of shortcuts. Hence while being a shortest path is a global property, we can verify it locally by checking a single face at a time for the presence of shortcuts adjacent to the path, ignoring the rest of the graph. Notice that this is only possible when the input graph is the union of non-crossing shortest paths, not for general plane graphs. Without this property, finding one distance is as difficult as finding a shortest path on U; where we recall that U is the graph arising from the union of all non-crossing shortest paths.

Shortcuts allow us to compute the lengths of non-crossing shortest paths in total linear time. Thus we extend the result in [21] also in the case in which U contains cycles. Our novel simple technique does not require the result by Gabow and Tarjan [11]. Moreover, we also propose an algorithm for listing the sequence of edges in a shortest path p joining a terminal pair in $O(\max\{\ell, \ell \log\log(\frac{k}{\ell})\})$, where ℓ is the number of edges in p.

In this way we prove that if there exists an algorithm able to compute the union of non-crossing shortest paths whose extremal vertices lie on the same face of a plane undirected graph, then we can compute the lengths of these paths in the same time complexity.

The algorithm we propose can be easily implemented and it does not require sophisticated data structures. A preliminary version of this algorithm can be found in [6]. We follow the same approach of Polishchuck and Mitchell [19], that was inspired by Papadopoulou's work [17]. They solve the problem of finding k non-crossing shortest paths in a polygon with n vertices, where distances are defined according to the Euclidean metric.

Structure The article is organized as follows: in Section 2 we give preliminary definitions and notations that will be used in the whole article. In Section 3 we deal with shortcuts and in Section 4 we use shortcuts in algorithm `ImplicitPaths` for describing an implicit representation of non-crossing shortest paths. This representation is used to compute distances between terminal pairs in linear time, and, in Section 5, it is also used to list the non-crossing shortest paths. Finally, in Section 6 conclusions are given and we mention some open problems.

2. Preliminaries

General definitions and notations are given. Then we deal with paths, non-crossing paths, and we define a partial order on terminal pairs, the *genealogy tree*. All graphs in this article are undirected.

2.1. Definitions and notations

Let $G = (V(G), E(G))$ be a plane graph, i.e., a planar graph with a fixed planar embedding. We denote by F_G the set of faces and by f_G^∞ its infinite face. When no confusions arise we use the term face to denote both the boundary and the finite region bounded by the boundary, and the infinite face is simply denoted by f^∞.

We use standard union and intersection operators on graphs.

Definition 1. Given two graphs $G = (V(G), E(G))$ and $H = (V(H), E(H))$, we define the following operations and relations:

- $G \cup H = (V(G) \cup V(H), E(G) \cup E(H))$;
- $G \cap H = (V(G) \cap V(H), E(G) \cap E(H))$;
- $H \subseteq G \iff V(H) \subseteq V(G) \wedge E(H) \subseteq E(G)$;
- $G \setminus H = (V(G), E(G) \setminus E(H))$.

Given a graph $G = (V(G), E(G))$, an edge e and a vertex v we write, for short, $e \in G$ in place of $e \in E(G)$ and $v \in G$ in place of $v \in V(G)$. An *ab path* is a path whose extremal vertices are a and b.

We use round brackets to denote ordered sets. For example, $\{a, b, c\} = \{c, a, b\}$ and $(a, b, c) \neq (c, a, b)$. Moreover, for every $r \in \mathbb{N}$ we denote by $[r]$ the set $\{1, \ldots, r\}$.

Let $\omega : E(G) \to \mathbb{R}^+$ be a weight function on edges. The weight function is extended to a subgraph H of G so that $\omega(H) = \sum_{e \in E(H)} \omega(e)$.

We assume that the input of our problem is a plane undirected graph $U = \bigcup_{i \in [k]} p_i$, where p_i is a shortest $x_i y_i$ path in a plane graph G, and the terminal pairs $\{(x_i, y_i)\}_{i \in [k]}$ lie on the infinite face f^∞ of G. We stress that we work with a fixed embedding of U. W.l.o.g. we assume that U is connected, otherwise it suffices to work on each connected component.

For a (possibly not simple) cycle C, we define the *region bounded by* C the maximal subgraph of U whose infinite face has C as boundary. If R is a subgraph of U, then we denote by ∂R the infinite face of R. Moreover, we define $\mathring{R} = R \setminus \partial R$.

Given a path p and a face f, we say that f is *adjacent* to p if p and f share at least one edge.

Let γ_i be the path in f^∞ that goes clockwise from x_i to y_i, for $i \in [k]$. We assume also that pairs $\{(x_i, y_i)\}_{i \in [k]}$ are *well-formed*, i.e., for all $j, \ell \in [k]$ either $\gamma_j \subseteq \gamma_\ell$ or $\gamma_j \supseteq \gamma_\ell$ or γ_j and γ_ℓ share no edges. We note that if terminal pairs are well-formed, then there exists a set of pairwise non-crossing shortest $x_i y_i$ paths. The reverse is not true if some paths are subpaths of the infinite face of G; this case is not interesting in the applications and has never been studied in literature, where the terminal pairs are always assumed to be well-formed. The well-formed property can be easily verified in linear time, since it corresponds to checking that a string of parentheses is balanced, and it can be done by a sequential scan of the string. We also assume that the terminal pairs are distinct, i.e., there does not exist any pair $i, j \in [k]$ such that $\{x_i, y_i\} = \{x_j, y_j\}$.

Given $i \in [k]$, we denote by *i-path* an $x_i y_i$ path. It is always useful to see each i-path as oriented from x_i to y_i, for $i \in [k]$, even if the path is undirected. For an i-path p, we define Left_p as the left portion of U with respect to p, i.e., the finite region bounded by the cycle formed by p and γ_i; similarly, we define Right_p as the right portion of U with respect to p, i.e., the finite region bounded by the cycle formed by p and $f^\infty \setminus \gamma_i$.

For an i-path p and a j-path q, we say that q *is to the right of* p if $q \subseteq \text{Right}_p$, similarly, we say that q *is to the left of* p if $q \subseteq \text{Left}_p$. Given $R \subseteq U$ and an i-path $p \subseteq R$, for some $i \in [k]$, we say that p is the *leftmost i-path in* R if p is to the left of q for each i-path $q \subseteq R$. Similarly, we say that p is the *rightmost i-path in* R if p is to the right of q for each i-path $q \subseteq R$.

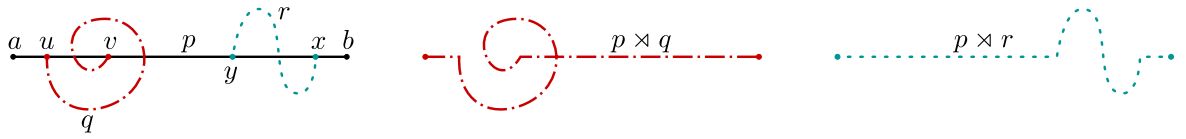

Fig. 3. Illustrating operator $\rtimes$.

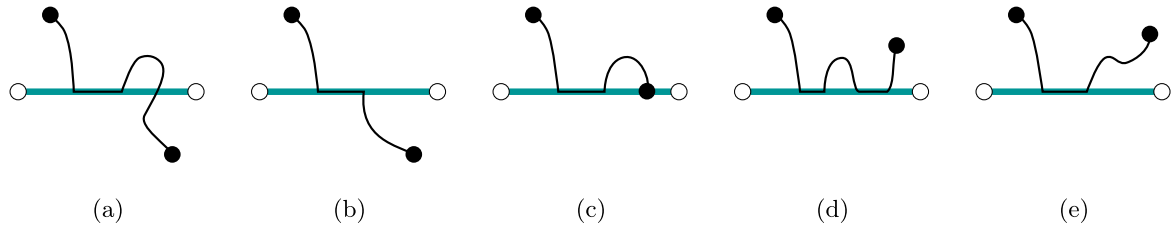

Fig. 4. Paths in (a) and (b) are crossing, while paths in (c), (d), (e) are non-crossing. Moreover, paths in (a), (c) and (d) are not single-touch, while paths in (b) and (e) are single-touch.

2.2. Paths and non-crossing paths

Given an ab path p and a bc path q, we define $p \circ q$ as the (possibly not simple) path obtained by the concatenation of p and q. Given a simple path p and two vertices u, v of p, we denote by $p[u, v]$ the subpath of p with extremal vertices u and v.

Now we introduce the operator $\rtimes$, explained in Fig. 3, that allows us to replace a subpath in a path.

Definition 2. Let p be a simple ab path, let $u, v \in V(p)$ such that a, u, v, b appear in this order in p and let q be a uv path. We denote by $p \rtimes q$ the (possibly not simple) path $p[a, u] \circ q \circ p[v, b]$.

We say that two paths in a plane graph G are *non-crossing* if the curves they describe in the graph embedding do not cross each other; a combinatorial definition of non-crossing paths can be based on the *Heffter–Edmonds–Ringel rotation principle* [18]. We stress that this property depends on the embedding of the graph. Non-crossing paths may share vertices and/or edges. We also define a class of paths that will be used later.

Definition 3. Two paths p and q are *single-touch* if $p \cap q$ is a (possibly empty) path.

Examples of non-crossing paths and single-touch paths are given in Fig. 4.

Our algorithm builds a set of single-touch paths even if the shortest p_i's paths in G composing the input graph $U = \bigcup_{i\in[k]} p_i$ are not pairwise single-touch. This may happen if there are more shortest paths in G joining the same pair of vertices. Uniqueness of shortest paths can be easily ensured by introducing small perturbations in the weight function of G. We wish to point out that the technique we describe in this article does not rely on perturbation, but we break ties by choosing rightmost or leftmost paths. This implies that our results can also be used in the unweighted case, as done in [3]. Note that the single-touch property does not depend on the embedding, and if the terminal-pairs are well-formed, then it implies the non-crossing property. This is explained in the following remark, and, for this reason, we can say that the solution found by our algorithm holds for any feasible planar embedding of the graph.

Remark 1. If $\{\pi_i\}_{i\in[k]}$ is a set of simple single-touch paths, where π_i is an i-path, for $i \in [k]$, then $\{\pi_i\}_{i\in[k]}$ is a set of pairwise non-crossing paths for all the embeddings of U such that the terminal pairs $\{(x_i, y_i)\}_{i\in[k]}$ are well-formed.

2.3. Genealogy tree

Given a well-formed set of pairs $\{(x_i, y_i)\}_{i\in[k]}$, we define here a partial ordering as in [21] that represents the inclusion relation between γ_i's. This relation intuitively corresponds to an *adjacency* relation between non-crossing shortest paths joining each pair.

Choose an arbitrary i^* such that there are neither x_j nor y_j, with $j \neq i^*$, walking on f^∞ from x_{i^*} to y_{i^*} (either clockwise or counterclockwise), and let e^* be an arbitrary edge on that walk. For each $j \in [k]$, we can assume that $e^* \notin \gamma_j$, indeed if it is not true, then it suffices to switch x_j and y_j. We say that $i \preceq j$ if $\gamma_i \subseteq \gamma_j$. We define the *genealogy tree* T_g of a set of well formed terminal pairs as the transitive reduction of poset $([k], \preceq)$.

Fig. 5 shows an example of well-formed terminal pairs, and the corresponding genealogy tree for $i^* = 1$. From now on, in all figures we draw f^∞ by a solid light grey line.

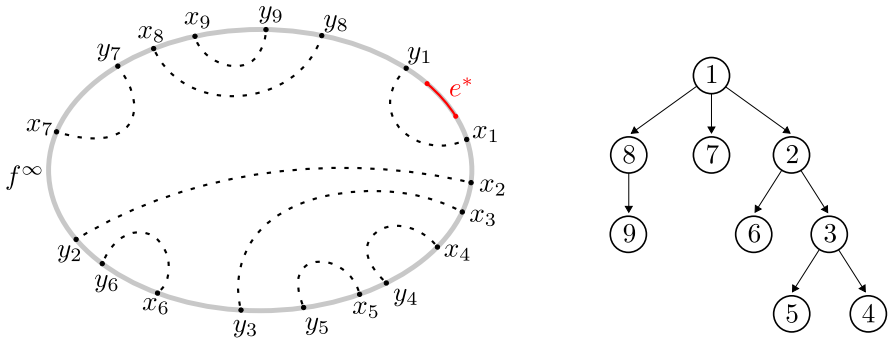

Fig. 5. On the left a set of well-formed terminal pairs. If we choose $i^* = 1$, then we obtain the genealogy tree on the right.

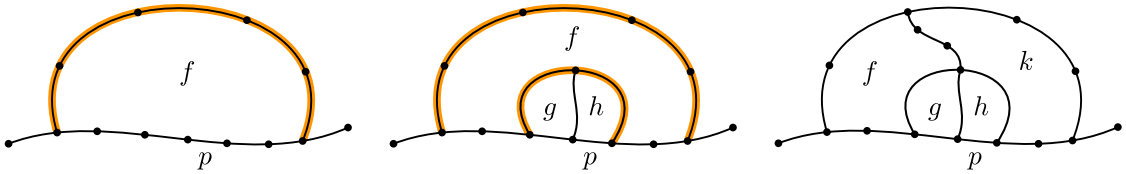

Fig. 6. All edges have unit weight. On the left, highlighted in orange, there is a shortcut for p contained in ∂f. In the middle there are two shortcuts for p both contained in ∂f. On the right there are no shortcuts for p. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

3. Shortcuts

Now we introduce *shortcuts*, that are the main tool of algorithm `ImplicitPaths` introduced in Section 4, and the most important theoretical novelty of this article.

Roughly speaking, a shortcut appears if there exists a face f adjacent to a path p so that we can modify p going around f without increasing its length. We show that we can decide whether a path is a shortest path by looking at the existence of shortcuts: in this way, we can check a global property of a path p—i.e., being a shortest path—by checking a local property—i.e., the presence of shortcuts in faces adjacent to p. This result is not true for general plane graphs, but it only holds when the input graph is the union of shortest paths joining well-formed terminal pairs on the same face. Shortcuts are the main tool of algorithm `ImplicitPaths` described in Section 4, and the most important theoretical novelty of this article.

Now we can formally define shortcuts, which are clarified in Fig. 6. The main application of shortcuts is stated in Theorem 1.

Definition 4. Given a path p and a face f containing two vertices $u, v \in p$, we say that a uv subpath q of ∂f not contained in p is a *shortcut for* p if $\omega(p \rtimes q) \leq \omega(p)$.

Theorem 1. *Let λ be an i-path, for some $i \in [k]$. If there are no shortcuts for λ, then λ is a shortest i-path.*

Proof. If $\lambda = p_i$ then the thesis holds. Thus let us assume by contradiction that $\omega(p_i) < \omega(\lambda)$ and λ has no shortcuts.

Let $a, b \in V(\lambda) \cap V(p_i)$ be two vertices such that $p_i[a, b]$ and $\lambda[b, a]$ share no edges (such a and b exist because $p_i \neq \lambda$ and they are both i-path). Let C be the simple cycle $p_i[a, b] \circ \lambda[b, a]$, and let R be the region bounded by C. If R is a face of U, then $p_i[a, b]$ is a shortcut for λ, absurdum. Hence we assume that there exist edges in $\mathring{R}$, see Fig. 7 on the left.

Either $R \subseteq \text{Left}_{p_i}$ or $R \subseteq \text{Right}_{p_i}$. W.l.o.g., we assume that $R \subseteq \text{Left}_{p_i}$. Being $U = \bigcup_{j \in [k]} p_j$, for every edge $e \in \mathring{R}$ there exists at least one path $q \in P$ such that $e \in q$. Moreover, the extremal vertices of q are in γ_i because paths in P are non-crossing and $R \subseteq \text{Left}_{p_i}$.

Now we show by construction that there exist a path $p \in P$ and a face f such that $f \subseteq R$, ∂f intersects λ on vertices and $\partial f \setminus \lambda \subseteq p$; thus $\partial f \setminus \lambda$ is a shortcut for λ because p is a shortest path.

For all $q \in P$ such that $q \subseteq \text{Left}_{p_i}$ we assume that $\text{Left}_q \subseteq \text{Left}_{p_i}$ (if it is not true, then it suffices to switch the extremal vertices of q).

For each $q \subseteq \text{Left}_{p_i}$, let $F_q = \{f \in F \mid \partial f \subseteq R \cap \text{Left}_q\}$, where F is the set of the faces of U. To complete the proof, we have to find a path p such that $|F_p| = 1$, indeed, the unique face f in F_p satisfies $\partial f \setminus \lambda \subseteq p$, and thus $\partial f \setminus \lambda$ is a shortcut for λ.

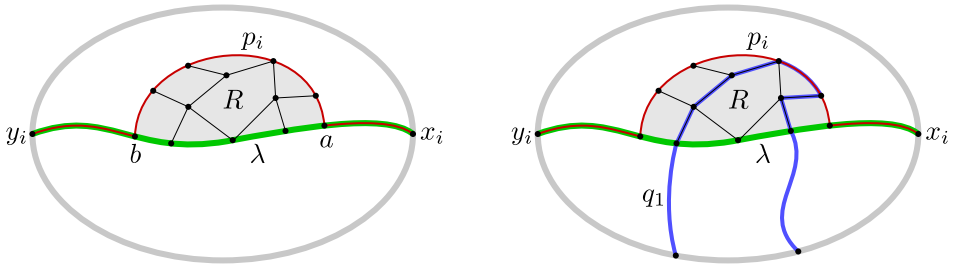

Fig. 7. Paths and regions used in Theorem 1's proof. Path λ is in green, p_i in red, q_1 in blue and region R is highlighted in grey. It holds that $|F_{q_1}| = 4$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Now, let $e_1 \in \mathring{R}$ and let $q_1 \in P$ be such that $e_1 \in q_1$. Being $e_1 \in \mathring{R}$, then $|F_{q_1}| < |F_{p_i}|$ and $|F_{q_1}| > 0$ because $e_1 \in q_1$, see Fig. 7 on the right. If $|F_{q_1}| = 1$, the proof is completed, otherwise we choose $e_2 \in \mathring{R} \cap \mathring{\text{Left}}_{q_1}$ and $q_2 \in P$ such that $e_2 \in q_2$. It holds that $|F_{q_2}| < |F_{q_1}|$ and $|F_{q_2}| > 0$ because $e_2 \in q_2$. By repeating this reasoning, and being $U = \bigcup_{j\in[k]} p_j$, we find a path p such that $|F_p| = 1$. □

Given a path p, we say that a path q is a *right shortcut for* p if q is a shortcut for p and $q \subseteq \text{Right}_p$. The following corollary can be proved by the same approach of Theorem 1 and is more useful for our purposes.

Corollary 1. *Let λ be an i-path, for some $i \in [k]$. If there are no right shortcuts for λ, then there does not exist any path $\lambda' \subseteq \text{Right}_\lambda$ satisfying $\omega(\lambda') \leq \omega(\lambda)$.*

4. Computing lengths in linear time

In Theorem 2 we show that the distances between terminal pairs can be computed in $O(|E(U)|)$ time by knowing U. This is the main result of this article. To achieve it, we introduce algorithm `ImplicitPaths`, that gives us an implicit representation of non-crossing shortest paths used in the proof of Theorem 2. The implicit representation is described in Remark 3.

The main idea behind algorithm `ImplicitPaths` is the following. We build a set of shortest i-paths $\{\lambda_i\}_{i\in[k]}$, by finding λ_i at iteration i, where the terminal pairs are numbered according to a postorder visit of T_g. In particular, at iteration i we find the rightmost shortest i-path in $U_i = \bigcap_{j\in[i-1]} \text{Right}_{\lambda_j}$ in the following way: first we set λ_i as the leftmost i-path in U_i, then we update λ_i by moving right through right shortcuts (the order in which shortcuts are chosen is not relevant). When λ_i has no more right shortcuts, then it is the rightmost shortest i-path in U_i by Corollary 1.

Algorithm `ImplicitPaths`:

Input: an undirected plane graph U composed by the union of k non-crossing shortest paths in a plane graph G each one joining a terminal pair on the infinite face of G

Output: an implicit representation of a set of paths $\{\lambda_1, \ldots, \lambda_k\}$, where λ_i is a shortest i-path, for $i \in [k]$

1 Compute T_g and renumber the terminal pairs according to a postorder visit of T_g;
2 **for** $i = 1, \ldots, k$ **do**
3 Let λ_i be the leftmost i-path in $U_i = \bigcap_{j\in[i-1]} \text{Right}_{\lambda_j}$ ($U_1 = U$);
4 **while** *there exists a right shortcut τ for λ_i in U_i* **do**
5 $\lambda_i := \lambda_i \rtimes \tau$;

Lemma 1. *Let $\{\lambda_i\}_{i\in[k]}$ be the set of paths given by algorithm `ImplicitPaths`. Then*

1.(1) λ_i is the rightmost shortest i-path in U_i, for $i \in [k]$;
1.(2) $\{\lambda_i\}_{i\in[k]}$ is a set of single-touch paths.

Proof. We proceed by induction to prove the first statement. Trivially λ_1 is the rightmost shortest 1-path in $U_1 = U$ because of Corollary 1. Let us assume that λ_j is the rightmost shortest j-path in U_j, for $j \in [i-1]$, we have to prove that λ_i is the rightmost shortest i-path in U_i.

In Line 3, we initialize λ_i as the leftmost i-path in U_i. By induction and the postorder visit, at this step, there does not exist in U_i any i-path p to the left of λ_i shorter than λ_i. Otherwise λ_i would cross a path λ_j, for some $j < i$, implying that λ_j is not a shortest j-path. We conclude by the while cycle in Line 4 and Corollary 1.

Statement *1.(2)* follows from *1.(1)*; indeed, if λ_i and λ_j are not single-touch, for some $i, j \in [k]$, then *1.(1)* is denied either for λ_i or for λ_j. □

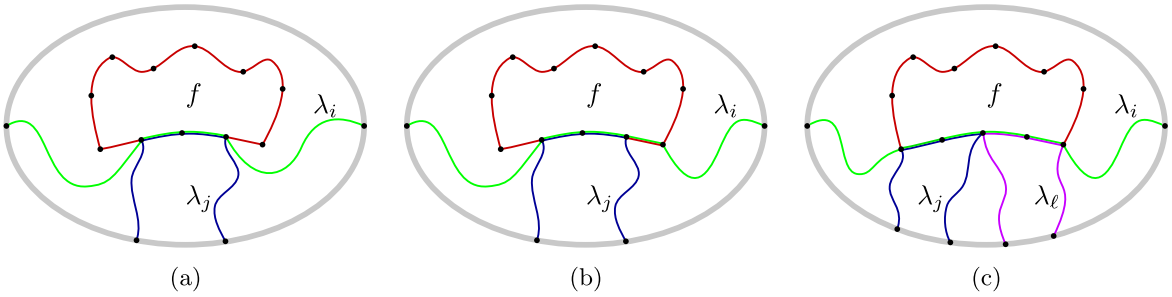

Fig. 8. If $C_i = \{\lambda_j, \lambda_\ell\}$, then in (a) there are no shortcuts for λ_i in f, while in (b) and (c) there may be.

Given $i \in [k]$ we define C_i as the set of children of i in the genealogy tree, moreover, we say that λ_j is a *child of* λ_i if $j \in C_i$.

Before stating our main result, we introduce a trivial consequence of non-crossing property and Jordan's Curve Theorem [13], indeed, every i-path π satisfies that $\pi \circ \gamma_i$ is a closed curve.

Remark 2. Let $\{\pi_i\}_{i\in[k]}$ be a set of non-crossing i-paths. Let $i, j, \ell \in [k]$. If π_i, π_j and π_ℓ share a common edge, then at least two among $\{i, j, \ell\}$ are a couple of ancestor/descendant in the genealogy tree.

Theorem 2. *Given an undirected plane graph U composed by the union of k non-crossing shortest paths in a plane graph G each one joining a terminal pair on the infinite face of G, we can compute the length of each shortest path in $O(|E(U)|)$ total time.*

Proof. We show that during the execution of algorithm `ImplicitPaths` we can also compute the length of λ_i, for all $i \in [k]$, in linear total time. If we also prove that algorithm `ImplicitPaths` can be executed in linear time, then the thesis follows from Lemma 1.

We define, for all $i \in [k]$, $\lambda_{i,0}$ as λ_i after Line 3, i.e., the leftmost i-path in $U_i = \bigcap_{j\in[i-1]} \text{Right}_{\lambda_j}$. We have to show that all the $\lambda_{i,0}$'s and all the shortcuts required in Line 4 can be computed in total linear time.

Let us start dealing with the shortcuts. We do a linear preprocessing that visits clockwise every face. Fixed a face f, let $v_1^f, v_2^f, \ldots, v_q^f$ be its vertices in clockwise order. We visit these vertices clockwise and compute the distance between v_1^f and v_i^f, for $i \in [q]$, and the length of the boundary of f. After this preprocessing, the lengths of the clockwise path and of the counterclockwise path in ∂f joining any given pair of vertices in f can both be computed in constant time. Now, if the intersection between ∂f and λ_i, for some $i \in [k]$, is contained in λ_j, for some $j \in C_i$, then we know that there are no right shortcuts in f for λ_i, otherwise they would be right shortcuts for λ_j, see Fig. 8.(a). Thus we ask for a right shortcut in f for λ_i if and only if λ_i visits at least one edge in ∂f that is not contained in its children, see Fig. 8.(b), or $\lambda_i \cap \partial f$ is contained in an least two children of λ_i, see Fig. 8.(a), and consequently at least one more edge of ∂f is visited. In this way, during the execution of algorithm `ImplicitPaths` we ask for a shortcut in f at most $O(|E(f)|)$ times thank also to Remark 2. This implies that finding all the shortcuts requires total linear time.

Now we prove that all the $\lambda_{i,0}$'s can be computed in total linear time. We stress that all the λ_i's and all the $\lambda_{i,0}$'s are represented as list, in this way we can join two paths in constant time. We recall that $\lambda_{i,0}$ is the leftmost i-path in $U_i = \bigcap_{j\in[i-1]} \text{Right}_{\lambda_j}$, C_i is the set of children of i, and γ_i is the clockwise path on the infinite face of G from x_i to y_i. Let $Y_i = \bigcup_{j\in C_i} \lambda_j$ and let f_i be the infinite face of $Y_i \cup \gamma_i$. We observe that, by its definition, $\lambda_{i,0}$ is the counterclockwise i-path on f_i. Clearly, if all the λ_j's, for $j \in C_i$, are vertex disjoint, then λ_j is contained in f_i, for all $j \in C_i$, see Fig. 9 on the left. If the λ_j's are not vertex disjoint, then some edges of Y_i are not in f_i, and by construction, they are not in f_ℓ, for all $i \preceq \ell$, see Fig. 9 on the right. Thus we can see the sequence of the f_i's as an updating graph for which if an edge is deleted at iteration i, then it does not appear again in f_ℓ for all $i \preceq \ell$. Hence, thanks to Remark 2, every edge appears at most two times in this construction. Consequently, all the $\lambda_{i,0}$'s can be computed in total linear time because also to their list representation.

We have proved that algorithm `ImplicitPaths` requires linear time. We use the same argument to compute paths' lengths. Let $i \in [k]$ and $j \in C_i$. At iteration i we know $\omega(\lambda_j)$, and we compute $\omega(\lambda_{i,0} \cap \lambda_j)$ by subtracting from $\omega(\lambda_j)$ the length of edges of λ_j that are not in $\lambda_{i,0}$. In this way we can compute the lengths of $\lambda_{i,0}$ for all $i \in [k]$ in total linear time because every edge is considered at most two times thanks to Remark 2. Being the shortcuts computable in linear time, the thesis follows. □

By following Theorem 2's proof we obtain the following implicit representation of the λ_i's.

Remark 3. Paths λ_i's computed by algorithm `ImplicitPaths` are implicitly represented as follows: $\lambda_i = q_1 \circ \lambda_{j_1}[a_1, b_1] \circ q_2 \circ \lambda_{j_2}[a_2, b_2] \circ \ldots \circ q_r \circ \lambda_{j_r}[a_r, b_r] \circ q_{r+1}$ where $\{j_1, j_2, \ldots, j_r\} \subseteq C_i$ and $E(q_\ell \cap \lambda_z) = \emptyset$ for all $\ell \in [r+1]$ and $z \in C_i$. Note that the q_ℓ's can be empty.

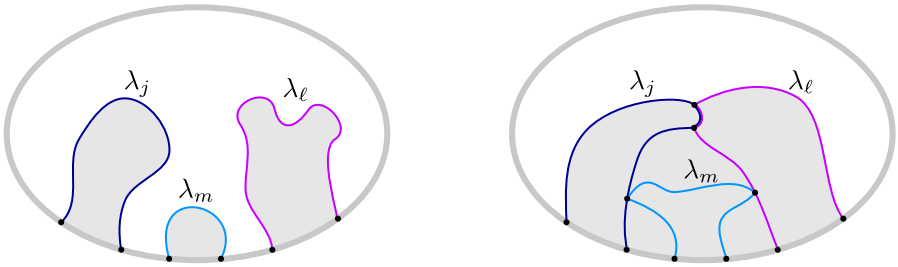

Fig. 9. λ_j, λ_ℓ and λ_m are the children of λ_i. The grey region represents f_i.

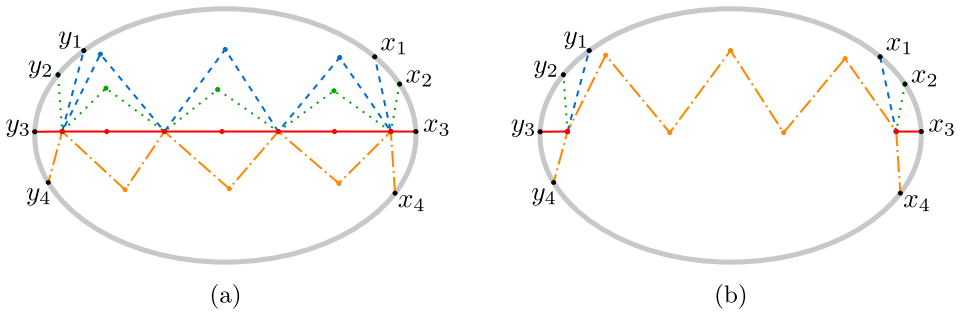

Fig. 10. (a) the union of shortest i-paths, for $i \in [4]$, in unit-weighted graph, each different path has different style, (b) the union of $\{\lambda_i\}_{i\in[4]}$, the output paths of algorithm `ImplicitPaths`.

Now we explain the implicit representation of the λ_i's. If i is a leaf of the genealogy tree, then λ_i is given explicitly. Otherwise we give explicitly edges that do not belong to the children of λ_i, that is the q_i's paths, and we give the intersection path between λ_i and one of its child by specifying the extremal vertices of this intersection. This representation requires linear space thanks to Remark 2.

5. Listing paths

We study the problem of listing the edges in λ_i, for some $i \in [k]$, after the execution of algorithm `ImplicitPaths`. We want to underline the importance of the single-touch property. In Fig. 10, in (a) four shortest paths are drawn (the graph is unit-weighted), we observe that the single-touch property is clearly not satisfied. A single-touch version of the previous four paths is drawn in (b); it can be obtained by algorithm `ImplicitPaths`. It is clear that the problem of listing the edges in a path in this second case is easier. We stress that in the general case the union of a set of single-touch paths can form cycles, see Fig. 1 for an example.

Theorem 3. *After $O(n)$ time preprocessing, each shortest path λ_i, for $i \in [k]$, can be listed in $O(\max\{\ell_i, \ell_i \log\log(\frac{k}{\ell_i})\})$ time, where ℓ_i is the number of edges of λ_i.*

Proof. For any $i \in [k]$, we denote by $\overrightarrow{\lambda_i}$ the oriented version of λ_i from x_i to y_i. During the execution of algorithm `ImplicitPaths`, we introduce a function `Mark` that marks a dart d with i if and only if the d is used for the first time in the execution of algorithm `ImplicitPaths` at iteration i. It means that $\texttt{Mark}(d) = i$ if and only if d belongs to $\overrightarrow{\lambda_i}$ and d does not belong to $\overrightarrow{\lambda_j}$, for all $j \preceq i$ and $j \neq i$. This function can be executed within the same time bound of algorithm `ImplicitPaths`. Now we explain how to find darts in $\overrightarrow{\lambda_i}$.

Let us assume that $(d_1, \ldots, d_{\ell_i})$ is the ordered sequence of darts in $\overrightarrow{\lambda_i}$. Let $v = \text{head}[d_{j-1}]$, and let us assume that $deg(v) = r$ in the graph $\bigcup_{j\in[k]} \lambda_j$. We claim that if we know d_{j-1}, then we find d_j in $O(\log\log r)$ time. First we order the outgoing darts in v in clockwise order starting in d_{j-1}, thus let $\text{Out}_v = (g_1, \ldots, g_r)$ be this ordered set (this order is given by the embedding of the input plane graph). We observe that all darts in Out_v that are in Left_{λ_i} are in $\overrightarrow{\lambda_w}$ for some $w \leq i$, thus $\texttt{Mark}(d) \leq i$ for all $d \in \text{Out}_v \cap \text{Left}_{\lambda_i}$. Similarly, all darts in Out_v that are Right_{λ_i} are in $\overrightarrow{\lambda_z}$ for some $z \geq i$, thus $\texttt{Mark}(d) \geq i$ for all $d \in \text{Out}_v \cap \text{Right}_{\lambda_i}$. Using this observation, we have to find the unique $l \in [r]$ such that $\texttt{Mark}(g_l) \leq i$ and $\texttt{Mark}(g_{l+1}) > i$. This can be done in $O(\log\log r)$ by using a van Emde Boas tree [22].

Being the $\overrightarrow{\lambda_i}$'s pairwise single-touch, then $\sum_{v\in V(\lambda_i)} deg(v) \leq 2k$, where the equality holds if and only if every $\overrightarrow{\lambda_j}$, for $j \neq i$, intersects on vertices $\overrightarrow{\lambda_i}$ exactly two times, that is the maximum allowed by the single-touch property.

Finally, if $2k \leq \ell_i$, then we list $\overrightarrow{\lambda_i}$ in $O(\ell_i)$ because the searches of the correct darts do not require more than $O(k)$ time, otherwise we note that

$$\sum_{\substack{j=1,\dots,\ell \\ a_1+\cdots+a_\ell \leq 2k}} \log\log a_j \leq \ell \log\log\left(\frac{2k}{\ell}\right),$$

so the time complexity follows. □

6. Conclusions

In this article we extend the result of Takahashi et al. [21] by computing the lengths of non-crossing shortest paths in undirected plane graphs also in the general case when the union of shortest paths is not a forest. Moreover, we provide an algorithm for listing the sequence of edges of each path in $O(\max\{\ell, \ell \log\log(\frac{k}{\ell})\})$, where ℓ is the number of edges in the shortest path.

We also introduced shortcuts on non-crossing shortest paths in plane graphs. They are a useful tool of interest itself.

All results of this article can be easily applied in a geometric setting, where it is asked to search for paths in polygons instead of plane graphs. The same results can be extended to the case of terminal pairs lying on two distinct faces, by the same argument shown in [21].

We left open the problem of listing a shortest path in time proportional to its length and finding the union of non-crossing shortest paths joining k terminal pairs lying on the same face of a plane graph in $o(n \log k)$ time.

Data availability

No data was used for the research described in the article.

References

[1] G. Ausiello, P.G. Franciosa, I. Lari, A. Ribichini, Max flow vitality in general and st-planar graphs, Networks 74 (2019) 70–78.
[2] L. Balzotti, Non-crossing shortest paths are covered with exactly four forests, 2022, CoRR.
[3] L. Balzotti, P.G. Franciosa, Non-crossing shortest paths in undirected unweighted planar graphs in linear time, J. Graph Algorithms Appl. 26 (2022) 589–606.
[4] L. Balzotti, P.G. Franciosa, Non-crossing shortest paths in undirected unweighted planar graphs in linear time, in: Computer Science – Theory and Applications, CSR 2022, in: Lecture Notes in Computer Science, vol. 13296, 2022, pp. 77–95.
[5] L. Balzotti, P.G. Franciosa, How vulnerable is an undirected planar graph with respect to max flow, in: Algorithms and Complexity: 13th International Conference, CIAC 2023, June 13–16, 2023, Proceedings, Springer, 2023.
[6] L. Balzotti, P.G. Franciosa, Non-crossing shortest paths lengths in planar graphs in linear time, in: Algorithms and Complexity: 13th International Conference, CIAC 2023, June 13–16, 2023, Proceedings, Springer, 2023.
[7] S.N. Bhatt, F.T. Leighton, A framework for solving VLSI graph layout problems, J. Comput. System Sci. 28 (1984) 300–343.
[8] D. Das, E. Kipouridis, M.P. Gutenberg, C. Wulff-Nilsen, A simple algorithm for multiple-source shortest paths in planar digraphs, in: 5th Symposium on Simplicity in Algorithms, SOSA, Virtual Conference, 2022, SIAM, 2022, pp. 1–11.
[9] D. Eisenstat, P.N. Klein, Linear-time algorithms for max flow and multiple-source shortest paths in unit-weight planar graphs, in: Symposium on Theory of Computing Conference, STOC'13, ACM, 2013, pp. 735–744.
[10] J. Erickson, A. Nayyeri, Shortest non-crossing walks in the plane, in: Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, SIAM, 2011, 297–308.
[11] H.N. Gabow, R.E. Tarjan, A linear-time algorithm for a special case of disjoint set union, J. Comput. System Sci. 30 (1985) 209–221.
[12] M.R. Henzinger, P.N. Klein, S. Rao, S. Subramanian, Faster shortest-path algorithms for planar graphs, J. Comput. System Sci. 55 (1997) 3–23.
[13] C. Jordan, Cours d'analyse de l'École polytechnique, Vol. 1, Gauthier-Villars et fils, 1893.
[14] P.N. Klein, Multiple-source shortest paths in planar graphs, in: Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2005, pp. 146–155.
[15] F.T. Leighton, Complexity Issues in VLSI: Optimal Layouts for the Shuffle–Exchange Graph and Other Networks, MIT Press, 1983.
[16] F.T. Leighton, New lower bound techniques for VLSI, Math. Syst. Theory 17 (1984) 47–70.
[17] E. Papadopoulou, k-Pairs non-crossing shortest paths in a simple polygon, Internat. J. Comput. Geom. Appl. 9 (1999) 533–552.
[18] T. Pisanski, P. Potocnik, J. Chen, J.L. Gross, T.W. Tucker, A. Vince, D. Archdeacon, S. Negami, A.T. White, Topological graph theory, in: Handbook of Graph Theory, Discrete Mathematics and Its Applications, Chapman & Hall / Taylor & Francis, 2003, pp. 610–786.
[19] V. Polishchuk, J.S.B. Mitchell, Thick non-crossing paths and minimum-cost flows in polygonal domains, in: Proceedings of the 23rd ACM Symposium on Computational Geometry, ACM, 2007, pp. 56–65.
[20] J.H. Reif, Minimum s-t cut of a planar undirected network in $O(n \log^2(n))$ time, SIAM J. Comput. 12 (1983) 71–81.
[21] J. Takahashi, H. Suzuki, T. Nishizeki, Shortest noncrossing paths in plane graphs, Algorithmica 16 (1996) 339–357.
[22] P. van Emde Boas, Preserving order in a forest in less than logarithmic time and linear space, Inform. Process. Lett. 6 (1977) 80–82.