Õptimal Fault-Tolerant Reachability Labeling in Planar Graphs

SHIRI CHECHIK, Tel Aviv University, Israel
SHAY MOZES, Reichman University, Israel
OREN WEIMANN, University of Haifa, Israel

We show how to assign labels of size $\tilde{O}(1)$ to the vertices of a directed planar graph G, such that from the labels of any three vertices s, t, f we can deduce in $\tilde{O}(1)$ time whether t is reachable from s in the graph $G \setminus \{f\}$. Previously it was only known how to achieve $\tilde{O}(1)$ queries using a centralized $\tilde{O}(n)$ size oracle [SODA'21].

1 INTRODUCTION

Reachability is undeniably one of the most fundamental properties of graphs. Determining the reachability between all pairs of vertices in a directed graph can be done in $\tilde{O}(\min\{n^\omega, mn\})$ time [1].

Reachability oracles. A *reachability oracle* is a compact data structure that allows to efficiently answer reachability queries between any pair of vertices in the graph. Henzinger et al. [20] provided conditional lower bounds for combinatorial constructions of reachability oracles, showing that no non-trivial combinatorial reachability oracle constructions exist for general directed graphs.[2] Specifically, they proved that it is impossible to design a reachability oracle that simultaneously achieves $O(n^{3-\epsilon})$ preprocessing time and $O(n^{2-\epsilon})$ query time, for any $\epsilon > 0$.

Since non-trivial reachability oracles are not attainable for general graphs, efforts have been directed towards developing improved reachability oracles for specific graph families. Notably, graphs possessing separators of size $s(n)$ admit a straightforward reachability oracle of size $\tilde{O}(n \cdot s(n))$ and query time $\tilde{O}(s(n))$. Consequently, planar graphs (and more extensive graph classes such as H-minor free graphs) admit oracles of size $\tilde{O}(n^{1.5})$ and query time $\tilde{O}(\sqrt{n})$. In a groundbreaking result, Thorup [33] introduced a near-optimal reachability oracle for directed planar graphs with $\tilde{O}(n)$ space and $\tilde{O}(1)$ query time. Subsequently, Holm et al. [21] further improved this construction to a truly optimal oracle with $O(n)$ space and $O(1)$ query time.

Fault-tolerant reachability oracles. In real-world networks, the vulnerability to changes and failures is a significant concern, and effectively handling failures has become crucial in modern computing. This motivation has sparked extensive research on data structures that can operate robustly in the presence of failures. Specifically, a *fault-tolerant* reachability oracle is designed to handle queries of the form s, t, f and determines whether vertex t is reachable from vertex s in the graph $G \setminus \{f\}$ (the graph G with vertex f and all its incident edges removed).

Fault-tolerant reachability oracles have been studied extensively in general graphs, see e.g. [7, 12, 18, 19, 26, 35]. In planar graphs, one can leverage the more powerful fault-tolerant *distance* oracles. Baswana et al. [8] introduced a single-source fault-tolerant distance oracle with near-optimal $\tilde{O}(n)$ space and $\tilde{O}(1)$ query time. They further extended their construction to handle the all-pairs variant of the problem at the cost of an increased $\tilde{O}(n^{1.5})$ space and $\tilde{O}(\sqrt{n})$ query time. Subsequently, Charalampopoulos et al. [11] presented an improved fault-tolerant distance oracle for the all-pairs version in planar graphs. Their fault-tolerant oracle accommodates multiple failures; however, it comes with a polynomial trade-off between the size of the oracle and the query time

[1]The $\tilde{O}$ notation suppresses polylogarithmic factors, and $\omega < 2.373$ is the exponent of matrix multiplication.
[2]The term combinatorial is often referred to algorithms that do not utilize fast matrix multiplications.

This work was initiated during Dagstuhl Seminar 22461 on Dynamic Graph Algorithms.
SM and OW supported in part by Israel Science Foundation grant 810/21.

that may be less favorable compared to previous constructions. It is important to highlight that, in the context of the all-pairs version, the fault-tolerant oracles mentioned above have considerably worse bounds when compared to the best known distance oracles without faults for planar graphs. Finally, in a groundbreaking development, Italiano et al. [23] in SODA 2021 introduced a nearly optimal fault-tolerant reachability oracle for directed planar graphs. Their innovative approach finally achieved near-optimal $\tilde{O}(n)$ size, $\tilde{O}(n)$ construction time, and $\tilde{O}(1)$ query time.

Fault-tolerant reachability labeling. A more powerful concept than oracles is that of *labeling schemes*, where each vertex is assigned a compact label, and the objective is to answer queries based solely on the labels of the involved vertices. Labeling schemes have proven to be valuable in distributed settings where inferring properties like distances or reachability is advantageous using only local information such as the labels of the source and destination vertices. Such scenarios arise in communication networks or disaster-stricken areas, where communication with a centralized entity may be infeasible or impossible. Labeling schemes provide a natural and structured framework for studying the distribution of graph information. Various problems have been explored within this model, including adjacency [3–5, 9, 24, 31], distances [1, 6, 16, 17, 33], flows and connectivity [22, 25, 28], and Steiner tree [30]. See [32] for a survey.

In this paper, we focus on *fault-tolerant* reachability labeling in directed planar graphs. The objective is to assign a compact label to each vertex, such that given the labels of any three vertices s, t, f, one can efficiently determine whether t is reachable from s in the graph $G \setminus \{f\}$. Fault-tolerant labeling schemes (also called forbidden-set labeling schemes) were heavily studied for various problems (such as connectivity, distances, and routing) and on various graph families (see e.g. [1, 2, 6, 13–15, 32, 34]).

For reachability in planar graphs, the best previously known fault-tolerant labeling scheme was the one for fault-tolerant distances by [6] in which the label size is $\tilde{O}(n^{2/3})$. Is this the best possible? Ideally, the goal would be to devise a labeling scheme in which the sum of the label sizes is roughly equal to the size of the state-of-the-art oracle. However, achieving this goal is not always possible. For example, in [10], an almost optimal exact distance oracle is given for directed planar graphs (without faults) of size $O(n^{1+o(1)})$ and query time $\tilde{O}(1)$. On the other hand, it was shown in [17] that exact distance labels (even without faults) for planar graphs necessitate polynomial-sized labels regardless of the query time. Given this discrepancy between oracles and labeling schemes for distances in planar graphs, a natural question arises: does the same discrepancy exist for fault-tolerant reachability? In other words, is it possible to design a labeling scheme for directed planar graphs with label size $\tilde{O}(1)$ and query time $\tilde{O}(1)$? Or does a similar gap exist between fault-tolerant reachability oracles and labeling schemes, as in the case of distances?

Our results. We answer this question in the affirmative by providing a near optimal labeling scheme for fault-tolerant reachability in directed planar graphs with $\tilde{O}(1)$ label size and $\tilde{O}(1)$ query time.

To obtain a fault-tolerant reachability labeling scheme, one might hope to generalize existing labeling schemes for *undirected* graphs. Specifically, for undirected planar graphs, Abraham et al. [1] presented labels of size $\tilde{O}(1)$ that for any fixed $\epsilon > 0$, from the labels of vertices s, t, and the labels of a set F of failed vertices, can report in $\tilde{O}(|F|^2)$ time a $(1 + \epsilon)$-approximation of the shortest s-to-t path in the graph $G \setminus F$. One would hope to generalize this result to the directed case, even just settling for the seemingly easier task of reachability, and even for a single fault. This is particularly tempting because previous results for the failure-free case have successfully generalized from undirected to directed planar graphs. For example, Thorup [33] was able to convert his $(1 + \epsilon)$-distance oracle for undirected planar graphs to also work in the directed case by employing clever ideas. However, in the presence of failures, the task becomes significantly more challenging. In a nutshell, one of the challenges in adapting undirected results to the directed case in the context of fault-tolerant reachability labeling in planar graphs is the following. In many planar graph papers, including [33] and [1], the algorithms store distances from a vertex to other vertices in relevant separators. In Thorup's reachability oracle [33], the algorithm stores, for each vertex s and each relevant path separator, the first vertex on the path that is reachable from s and the last vertex on the path that can reach s. To determine if vertex s can reach vertex t by a path that intersects the path separator, one can examine the information stored in the labels of s and t. By utilizing this stored information, it becomes possible to check the reachability between s and t. One of the main challenges we face with this approach is the occurrence of failures anywhere along the relevant path separator. A faulty vertex f on the path separator requires us to store additional information, including the closest vertex to f that appears after f on the path separator and is reachable from the starting vertex s. Considering that failures can happen at any vertex on the path separator, this means we would need to store all vertices that are reachable from s and to s on the path separator. This requirement renders the approach impractical.

To address this issue, we need to adopt a different approach and develop new techniques specifically tailored for accommodating failures in the directed case.

It is worth mentioning that in both our labeling scheme and the oracle proposed in [23], the most challenging scenario arises when all three vertices s, t, and f are situated on the same path separator P. In [23], this situation was addressed by employing a complex data structure that extends dominator trees and previous data structures designed for handling strong-connectivity in general (non-planar) graphs under failures [19]. However, these data structures do not seem to be distributable into a labeling scheme (for example, they rely on an orthogonal range data structure and on a binary search step which do not seem suitable for a labeling scheme). We tackle this case without relying on dominator trees or similar sophisticated techniques. Therefore, our algorithm not only achieves the milestone of introducing a near optimal efficient fault-tolerant labeling scheme for planar reachability, but it also boasts a simpler approach compared to the one employed in [23]. We firmly believe that the simplicity of our algorithm makes it an important milestone in generalization to labels for approximate distances and for multiple failures in directed planar graphs and related graph families.

2 PRELIMINARIES

The decomposition tree. In [33, Lemma 2.2], Thorup proved that we can assume the graph G has an undirected spanning tree T (i.e., T is an unrooted spanning tree in the undirected graph obtained from G by ignoring the directions of edge) such that each path in T is the concatenation of $O(1)$ directed paths in G.

This way, we can describe the process of decomposing G into pieces in the undirected version of G. After describing the decomposition, we will replace each undirected path of T defined in the process by its $O(1)$ corresponding directed paths in G. We therefore proceed to describe the decomopostion treating G as an undirected graph with a rooted spanning tree T.

A balanced fundamental cycle separator [27, Lemma 5.3.2] (cf. [29] and [33, Lemma 2.3]) is a simple cycle C in G whose vertices are those of a single path of the (unrooted) spanning tree T, such that the removal of the vertices of C and their incident edges separates G into two roughly equal sized subgraphs. The balance of the separator can be defined with respect to a weight function on the vertices of G rather than just the number of vertices.

The recursive decomposition tree $\mathcal{T}$ of G is defined as follows. Each node of $\mathcal{T}$ corresponds to a subgraph of G (called a *piece*). The root piece of $\mathcal{T}$ is the entire graph G. The boundary ∂G of G is defined to be the empty set. We define the children of a piece H in $\mathcal{T}$ recursively. Let ∂H

be the boundary of H, and let C be a fundamental cycle separator which balances the number of non-boundary vertices of H. Let Q be the set of maximal subpaths of C that are internally disjoint from ∂H. To reduce clutter we sometimes refer to a vertex $v \in Q$, by which we mean that v belongs to some path in Q. The vertices of H that are enclosed by C (including the vertices of C) belong to one child H_1 of H. The vertices of subpaths in Q and the vertices of H not enclosed by C belong to the other child H_2. Note that the vertices of Q are the only vertices of H that belong to both children. The endpoints of Q that belong to ∂H are called the *apices* of H. The importance of apices arises from the fact that apices are the only vertices that belong to more than two pieces at the same depth of $\mathcal{T}$.[3] We call the paths in Q without the apices of H the *separator* of H. We do not include the apices in the separator paths to guarantee that the separator is vertex disjoint from ∂H. The boundary ∂H_i of a child H_i of H consists of the separator of H and of the subpaths of ∂H induced by the vertices of H_i.

Since H_2 might not contain all the vertices of C, the subgraph induced on the spanning tree T by H_2 may become disconnected. To overcome this slight technical issue we embed in H_2, if necessary, an artificial root connected by artificial edges to the rootmost apex in each of the resulting components of the tree. The embedding remains planar since all these apices were on the fundamental cycle separator of the parent piece H. We treat the artificial root and edges as part of the spanning tree of H_2. To guarantee that the addition of the artificial root and edges does not affect the reachability of non-artificial vertices of H_2, we direct the artificial edges into the artificial root.

The leaves of $\mathcal{T}$ (called *atomic* pieces) correspond to pieces with $O(1)$ non-boundary vertices. The depth of $\mathcal{T}$ is $O(\log n)$. For convenience, we consider all $O(1)$ vertices of an atomic (leaf) piece that are not already boundary vertices as the separator of the piece. It follows that the boundary ∂H of any piece H consists of $\tilde{O}(1)$ vertex disjoint paths (the subpaths induced by the vertices of H on the separators of the ancestor pieces of H), and each of the paths in ∂H lies on a single face of H. Also, since in each node of the decomposition tree only $\tilde{O}(1)$ apices are created, there are $\tilde{O}(1)$ apices along any root-to-leaf path in $\mathcal{T}$.

Having defined the decomposition tree $\mathcal{T}$ we can go back to treating G as a directed graph. As we explained above, each path we had discussed in the undirected version of G is the union of $O(1)$ directed paths in G. From now on when we refer to the separator paths of a piece H (resp., paths of ∂H), we mean the set of directed paths comprising the undirected separator paths of H (resp., the set of directed paths comprising the paths of ∂H).

To be able to control the size of the labels in our construction we need to be aware of the number of pieces of $\mathcal{T}$ to which a vertex belongs. The only vertices of a piece H that belong to both its children are the vertices of the separator path of H and the $\tilde{O}(1)$ apices of H. The fact that separator paths are disjoint from the boundary imply that every vertex belongs to the separator of at most one piece in $\mathcal{T}$. Hence, if a vertex is not an apex, it appears in $O(\log n)$ pieces of $\mathcal{T}$. Apices, on the other hand require special attention because they may belong to many (e.g., polynomially many) pieces of $\mathcal{T}$; High degree vertices may be apices in many pieces, and we will need a special mechanism for dealing with such vertices. Dealing with apices (like dealing with holes in other works on planar graphs) introduces technical complications that are not pertinent to understanding the main ideas of our work.[4]

[3]The term apex was previously used in exactly the same context in [1].

[4]A reader who is not interested in those details can safely skip the parts dealing with apices and just act under the assumption that each vertex appears in 2 atomic pieces (leaves) of $\mathcal{T}$, and that the following definition of ancestor pieces of a vertex v just degenerates to the set of $O(\log n)$ ancestors of the 2 atomic pieces containing v.

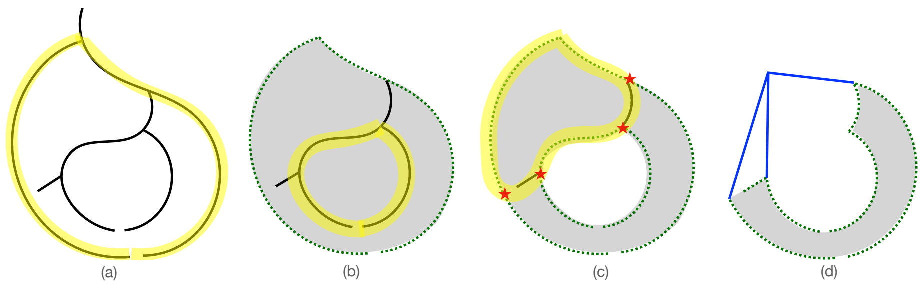


Fig. 1. Illustration of the recursive decomposition process. (a) a portion of the spanning tree T of G is shown in black. A balanced fundamental cycle separator of G is highlighted in yellow. (b) The piece H containing the vertices of G enclosed by the fundamental cycle of G is indicated in shaded grey. The boundary ∂H (dashed) consists of the separator of G. Portions of the spanning tree T induced by the vertices of H are shown, along with a fundamental cycle separator of H, highlighted in yellow. (c) The piece H_2 containing the vertices of H not strictly enclosed by the fundamental cycle separator of H is indicated in shaded grey. The boundary ∂H_2 (dashed) consists of ∂H and of the separator of H. Portions of the spanning tree T induced by the vertices of H_2 are shown, along with a fundamental cycle separator of H_2, highlighted in yellow. There are two maximal subpaths Q of the fundamental cycle separator of H_2 that do not belong to ∂H_2 (the two parts of the highlighted yellow cycle that are not dashed). The endpoints of Q that belong to ∂H_2 are the apices of H_2 (the four red stars). (d) The piece H_{22} containing the vertices of H_2 not strictly enclosed by the fundamental cycle in of H_2 is indicated in shaded grey. The boundary ∂H_{22} (dashed) consists of the paths induced by H_{22} on ∂H_2, and of the separator of H_2. Since the subgraph induced on T by the vertices of H_{22} is disconnected, artificial root and edges (in blue) are added to H_{22}.

We associate with every vertex $v \in G$ the (at most 2) rootmost pieces H in $\mathcal{T}$ in which v is an apex (or the atomic pieces containing v if v is never an apex). We denote these pieces by H_v. Note that every piece that contains a vertex v is either an ancestor of a piece in H_v or a descendant of a piece in H_v. For a vertex $v \in G$ we define the *ancestor pieces* of v to be the set of (weak) ancestors in $\mathcal{T}$ of the pieces H_v. By definition of H_v, every vertex, apex or not, has $O(\log n)$ ancestors pieces. We similarly define the ancestor separators/paths/apices of a vertex $v \in G$ as the separators/separator-paths/apices of any ancestor piece of v. See Figure 2.

We say that the separator Q of a piece H *separates* two vertices u and v (in H) if any u-to-v path in H must touch the separator Q of H or an apex of H. I.e., if at least one of the following holds: (1) $u \in Q$ or u is an apex of H, or (2) $v \in Q$ or v is an apex of H, or (3) each of u and v is in one distinct child of H. Note that if Q separates u and v in H then every u-to-v path in G either touches Q or touches the boundary of H.

For a subgraph H, a path P and a vertex v, let $\mathrm{first}_H(v, P)$ denote the first vertex of P that is reachable from v in H, and let $\mathrm{last}_H(v, P)$ denote the last vertex of P that can reach v in H. If vertex u appears before vertex v on a path P then we denote this by $u <_P v$ (or simply $u < v$ if P is clear from the context). Throughout the paper, we gradually describe the information stored in the labels along with the explanations of why this particular information is stored (and why it is polylogarithmic). To assist the reader, we highlight in gray the parts that describe the information stored. For starters, we let every vertex $v \in G$ store in its label, for every ancestor path P of v, the identity of P and, if $v \in P$, the location of v in P (so that given two vertices u, v of P, we can tell if $u < v$). We denote $P[u, v]$ the subpath of P between vertices u and v.

Thorup's non-faulty labeling. Using the above definitions and notations, it is now very simple to describe Thorup's non-faulty reachability labeling [33]. Consider any vertex v. Let H be the

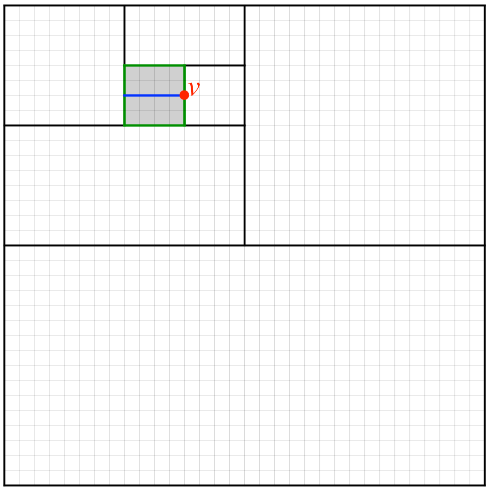


Fig. 2. Pieces in a recursive decomposition. The separators in this example alternate between horizontal and vertical lines. All rectangular pieces that contain the gray piece H are its ancestors. The boundary ∂H of H is shown in green. The maximal subpath of the cycle separator of H that is internally disjoint from ∂H is shown in blue. The vertex v is an apex of H because it is an endpoint of this maximal subpath. The separator of H is the blue path (without its endpoints).

rootmost piece in $\mathcal{T}$ in which v belongs to the separator. The crucial observation is that v is separated from every other vertex in G either by the separator of H or by the separator of some ancestor piece of H. Hence, every vertex $v \in G$ stores in its label $\text{first}_G(v, P)$ and $\text{last}_G(v, P)$ for every path P of the separator of every ancestor of the rootmost piece in which v belongs to the separator. Then, given a query pair u, v, there exists a u-to-v path in G if and only if $\text{first}_G(u, P) < \text{last}_G(v, P)$ for one of the $O(1)$ paths P of the separator of an ancestor piece of the rootmost piece whose separator separates u and v. Both u and v store the relevant information for these paths in their labels. We note that in Thorup's scheme we do not need to worry about apices since each vertex v only stores information in pieces above the first time v appears on a separator.

3 THE LABELING

In this section, we explain our labeling scheme. I.e., what to store in the labels so that given the labels of any three vertices s, f, t we can infer whether t is reachable from s in $G \setminus \{f\}$. We call the s-to-t path R in $G \setminus \{f\}$ the *replacement path*.

Let $\hat{H}$ be the rootmost piece in $\mathcal{T}$ whose separator Q separates t and f. Let H be a child piece of $\hat{H}$ that contains t (if both children of $\hat{H}$ contain t then, if one of the children does not contain f we choose H to be that child). Note that by choice of H, $f \notin H \setminus \partial H$. We assume without loss of generality that $s \in \hat{H}$. We handle the other case by storing a symmetric label to the one described here in the graph G with all edges reversed (the reverse graph of G has exactly the same decomposition tree as G, but there the roles of s and t are swapped, so the assumption does hold). Observe that by definition of H and of separation, $f \in \partial H$ iff $f \in Q$. In what follows, we separately handle the cases where $f \notin Q$ and $f \in Q$. The main challenge is in the latter case.

3.1 When $f \notin Q$ (and so, $f \notin \partial H$)

Consider first the case when the replacement path R does not touch ∂H. i.e., s, t and R are all contained in $H \setminus \partial H$. We handle this case by having each vertex $v \in H \setminus \partial H$ (and in particular s and t) store the standard (non-faulty) labeling of Thorup for $H \setminus \partial H$. Since each vertex $v \in G$ is non-boundary in $O(\log n)$ pieces of $\mathcal{T}$, this contributes $\tilde{O}(1)$ to the size of the label at each vertex.

We next consider the case that R touches ∂H. In this case, R must have a suffix contained in H, and this suffix is unaffected by the fault f. More precisely, R exists iff $\mathrm{first}_{G\setminus\{f\}}(s, P) < \mathrm{last}_H(t, P)$ for one of the paths P forming ∂H. It is easy to find $\mathrm{last}_H(t, P)$; For every vertex v in H, if H is an ancestor piece of v, then v stores in its label $\mathrm{last}_H(v, P)$ for each of the $O(\log n)$ paths P of ∂H. Since each vertex has only $O(\log n)$ ancestor pieces, this contributes $\tilde{O}(1)$ to the size of the label. Notice that by the rootmost choice of $\hat{H}$, H is an ancestor piece of t, so t indeed stores $\mathrm{last}_H(t, P)$. It thus remains only to describe how to find $\mathrm{first}_{G\setminus\{f\}}(s, P)$ from the labels of s and f.

For every vertex $s \in G$, for every ancestor apex f of s and for every ancestor path P of s, s stores in its label $\mathrm{first}_{G\setminus\{f\}}(s, P)$. Similarly, for every vertex $f \in G$, for every ancestor apex s of f and for every ancestor path P of f, f stores in its label $\mathrm{first}_{G\setminus\{f\}}(s, P)$.

If either s or f stores $\mathrm{first}_{G\setminus\{f\}}(s, P)$, we are done. Otherwise, consider the set of leafmost pieces in $\mathcal{T}$ that contain both s and f. Let H'' be such a piece. It must be that H'' is an ancestor piece of both s and f or else one of s and f is an ancestor apex of the other and stores $\mathrm{first}_{G\setminus\{f\}}(s, P)$. It follows that if neither s nor f store $\mathrm{first}_{G\setminus\{f\}}(s, P)$, then there are only $O(1)$ leafmost pieces that contain both s and f. To avoid unnecessary clutter we shall assume there is a unique piece H''. In reality we would have to apply the same argument for all $O(1)$ such pieces. Since H'' is an ancestor piece of both s and f, we can find the piece H'' by traversing the list of ancestors of s (stored in s) and of f (stored in f) until finding the lowest common ancestor. Let H' be the child piece of H'' that contains only s (if H'' is an atomic piece then define $H' = H''$). Recall that both s and f are in $\hat{H}$, so $\hat{H}$ is a (possibly weak) ancestor of H''.

Notice that if $\mathrm{first}_G(s, P) \neq \mathrm{first}_G(f, P)$ then the path in G from s to $\mathrm{first}_G(s, P)$ does not go through f and so $\mathrm{first}_{G\setminus\{f\}}(s, P) = \mathrm{first}_G(s, P)$. To check this, every vertex s in G, for each of the $O(\log n)$ ancestor paths P of s, stores $\mathrm{first}_G(s, P)$.

We handle the other case, where $\mathrm{first}_G(s, P) = \mathrm{first}_G(f, P)$ as follows. Consider a path in G from s to $\mathrm{first}_G(s, P)$. We can choose such a path that: (1) begins with a prefix that is contained in H' from s to $\mathrm{first}_{H'}(s, P')$ where P' is some path of $\partial H'$, then (2) continues along P' until the last vertex u of P' s.t $\mathrm{first}_G(u, P) = \mathrm{first}_G(f, P)$, and (3) ends with a suffix S in G from u to $\mathrm{first}_G(f, P)$. See Figure 3. Notice that the set of vertices $p \in P'$ s.t $\mathrm{first}_G(p, P) = \mathrm{first}_G(f, P)$ is a contiguous interval I of P' ending in u (this is because any vertex of P' earlier than u can reach $\mathrm{first}_G(f, P)$ through u). We think of all $p \in I$ as reaching $\mathrm{first}_G(u, P)$ in G using the same suffix S.

For every vertex f of G, for every pair of ancestor paths P', P of f, for the maximal interval I of vertices $p \in P'$ such that $\mathrm{first}_G(p, P) = \mathrm{first}_G(f, P)$, if the path S from the last vertex u of I to $\mathrm{first}_G(f, P)$[5] goes through f, we let f store the indices of the endpoints of I. Every vertex s of G stores $\mathrm{first}_A(s, P')$ for every ancestor piece A of s and every path P' of ∂A. This way, we can check if $\mathrm{first}_{H'}(s, P')$ is in the interval I (the interval stored by f for the pair of paths P', P). If $\mathrm{first}_{H'}(s, P')$ is not in the interval I, then the path in G from s to $\mathrm{first}_G(s, P)$ does not go through f and therefore $\mathrm{first}_{G\setminus\{f\}}(s, P) = \mathrm{first}_G(s, P)$ and we already have $\mathrm{first}_G(s, P)$ stored in s. Otherwise, $\mathrm{first}_{H'}(s, P')$ is in the interval I and the path in G from s to $\mathrm{first}_G(s, P)$ does go through f. Let S_1 be the prefix of S ending just before f, and let S_2 be the suffix of S starting immediately after f. Then, there are two options regarding the path in $G \setminus \{f\}$ from s to $\mathrm{first}_{G\setminus\{f\}}(s, P)$:

[5]If there are multiple paths in G from u to $\mathrm{first}_G(f, P)$ we fix and use an arbitrary such path.

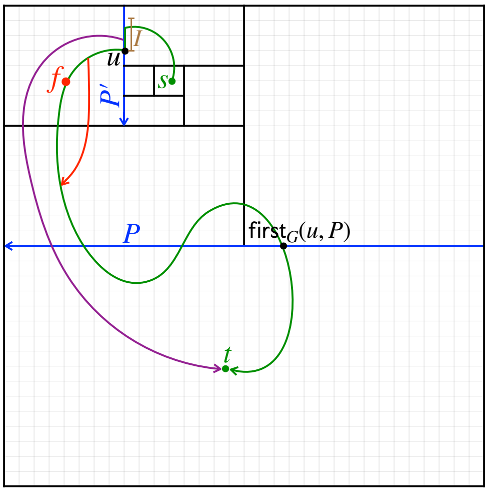


Fig. 3. When $f \notin Q$: The (green) s-to-t path in G first touches the (blue) separator path $P' \in \partial H'$, then continues along P' to u, then goes through f to $\text{first}_G(u,P)$ on the (blue) separator path $P \in \partial H$, and from there to t (without crossing P again). The (brown) interval I contains all vertices p that can reach $\text{first}_G(p,P)$ through u. When f fails, the replacement s-to-t path in $G \setminus \{f\}$ either takes a (red) detour around f (and then f stores this information) or is (purple) completely disjoint from the original (green) u-to-$\text{first}_G(u,P)$ part (and then s stores this information).

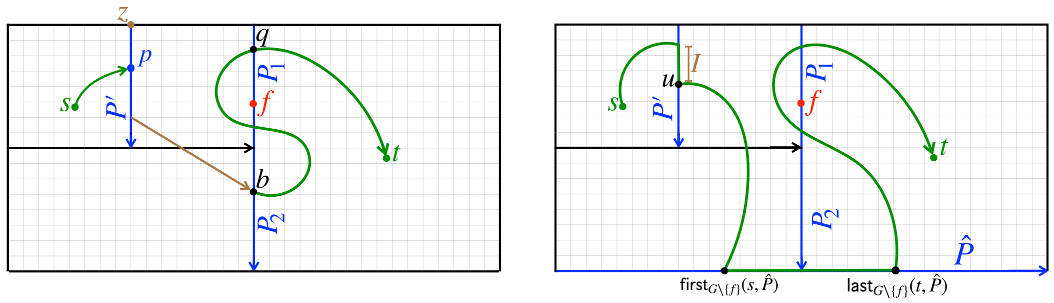
(1) The path intersects S_2. In this case, the path can continue along S_2 until reaching $\text{first}_G(f,P)$, so $\text{first}_{G\setminus\{f\}}(s,P) = \text{first}_G(f,P)$ and we have it stored in f. It only remains to check if this is indeed the case. Consider all vertices $p \in I$ that can reach $\text{first}_G(f,P)$ in $G \setminus \{f\}$. They must constitute a (possibly empty) prefix of I (since any such vertex $p \in I$ can reach any later vertex of I by going along I). We therefore let f store the index of the last vertex of the prefix of I that can reach $\text{first}_G(f,P)$ in $G \setminus \{f\}$. This way, we only need to check if $\text{first}_{H'}(s,P')$ (that is already stored in s) is earlier than this vertex.

(2) The path is disjoint from S_2. To handle this case, we will identify two valid candidates for $\text{first}_{G\setminus\{f\}}(s,P)$ and take the earlier in P of these two candidates:
   (a) The path is disjoint from S. To handle this case, for every vertex s of G, for every pair of ancestor paths P',P of s, for the path S defined by P', P and s[6], s stores $\text{first}_{G\setminus S}(s,P)$ (i.e., the first vertex of P that is reachable from s in G using a path that is disjoint from S). Since $f \in S$ then $f \notin G \setminus S$ and so $\text{first}_{G\setminus S}(s,P)$ is a valid candidate for $\text{first}_{G\setminus\{f\}}(s,P)$.
   (b) The path intersects S_1, so it might as well go through u (the last vertex of I). To handle this case, f stores $\text{first}_{G\setminus\{f\}}(u,P)$, which is also a valid candidate for $\text{first}_{G\setminus\{f\}}(s,P)$.

We summarize the case of $f \notin Q$ with a general corollary that follows from the above.

Corollary 3.1. *There is a (polylogarithmic-size) labeling to the vertices of G that returns* $\text{first}_{G\setminus\{f\}}(s,P)$ *given the labels of any two vertices s, f and the identity of any path P that (i) belongs to the boundary of an ancestor of H', and (ii) does not contain f. Symmetrically, we can find* $\text{last}_{G\setminus\{f\}}(t,P)$ *from the labels of any two vertices t, f.*

[6]To be clear, S is the path from the last vertex u of P' s.t. $\text{first}_G(u,P) = \text{first}_G(s,P)$ to $\text{first}_G(s,P)$



z
p
s
P'
q
P_1
f
t
b
P_2
u
I
s
P'
P_1
f
t
P_2
$\hat{P}$
$\text{first}_{G\setminus\{f\}}(s,\hat{P})$
$\text{last}_{G\setminus\{f\}}(t,\hat{P})$

Fig. 4. When $f \in Q$: In the left example, the (green) replacement path R touches only $P \in Q$ (the path on which f lies). In this case, b is the first vertex of P_2 that is reachable from s in a path internally disjoint from Q. The path from b to $q = \text{first}_{G\setminus\{f\}}(b, P_1)$ is then found using the mechanism of Section 4. In the right example, the (green) replacement path R touches some other path $\hat{P} \neq P$ (on which f does not lie). In this case, using Corollary 3.1 we check if $\text{first}_{G\setminus\{f\}}(s,\hat{P}) <_{\hat{P}} \text{last}_{G\setminus\{f\}}(t,\hat{P})$.

3.2 When $f \in Q$

Let P be the path of Q that contains f. Consider first the case where the replacement path R touches some path $\hat{P} \neq P$ of Q or of the boundary of some ancestor of H. Since boundary paths are vertex disjoint, $f \in P$ implies $f \notin \hat{P}$. Hence, in this case we can use Corollary 3.1 twice, once to obtain $\text{first}_{G\setminus\{f\}}(s,\hat{P})$, and once (using the symmetric part of Corollary 3.1) to obtain $\text{last}_{G\setminus\{f\}}(t,\hat{P})$. Then, s can reach t in $G \setminus \{f\}$ iff $\text{first}_{G\setminus\{f\}}(s,\hat{P}) <_{\hat{P}} \text{last}_{G\setminus\{f\}}(t,\hat{P})$. See Figure 4 (right). Therefore, in the remainder of this paper we deal with the case where other than P, R does not touch any path of Q or any path of the boundary of an ancestor of H. In this case, we cannot apply Corollary 3.1 since P contains f.

Let P_1 be the prefix of P ending just before f, and let P_2 be the suffix of P starting immediately after f. Recall that s is assumed to be in $\hat{H}$, the parent piece of H. Consider the replacement path R. Since R does not touch any path on the boundary of an ancestor of H, R is contained in $\hat{H}^\circ = \hat{H} \setminus \partial\hat{H}$. R starts with an s-to-b prefix that is internally disjoint from Q and $b = \text{first}_{\hat{H}^\circ\setminus Q}(s, P_1)$ or $b = \text{first}_{\hat{H}^\circ\setminus Q}(s, P_2)$. Our first goal is to find these vertices b. Finding $b_1 = \text{first}_{\hat{H}^\circ\setminus Q}(s, P_1)$ is easy since $b_1 = \text{first}_{\hat{H}^\circ\setminus Q}(s, P)$ (or else b_1 does not exist) so we let every vertex s in G store $\text{first}_{\hat{H}^\circ\setminus Q}(s, P)$ for each of the $O(1)$ paths P of each separator Q of each of the $O(\log n)$ ancestor pieces $\hat{H}$ of s. It therefore remains to find $b_2 = \text{first}_{\hat{H}^\circ\setminus Q}(s, P_2)$. We note that if $s \in P_1$ then we will not need b_2, and if $s \in P_2$ then $b_2 = s$.

If either s or f is an ancestor apex of the other, then we store b_2 explicitly in either s or f. That is, for every vertex s (resp. f) of G, for every ancestor apex f of s (resp. s of f), for every ancestor path P of s (resp. f), if f (resp. s) lies on P then s (reps. f) stores in its label $b_2 = \text{first}_{G\setminus\{f\}}(s, P_2)$, where P_2 is the suffix of P starting after f (resp. s).

Recall the definition of H'' and H' from Section 3.1. As in Section 3.1, since we handled the case that s or f are apices, we can assume that H'' and H' are uniquely defined. Consider the s-to-b_2 path in $\hat{H}^\circ \setminus Q$. It consists of a prefix that is internally disjoint from $\partial H'$ and ends at a vertex of one of the $O(1)$ paths $P' \in \partial H'$ (by definition of H', $f \notin H'$, so f does not belong to this prefix nor to P'). Let p be the first vertex of P' that is reachable in $\hat{H}^\circ$ from s with a path internally disjoint from $\partial H'$. Let z be the first vertex of P'. Let $P(z)$ denote the set of vertices of P that are reachable from z in $\hat{H}^\circ$ with paths that are internally disjoint from Q. Note that such paths in $\hat{H}^\circ \setminus Q$ from z to the vertices of $P(z)$ all enter P from the same side (since they cannot touch $\partial\hat{H}$ nor the separator

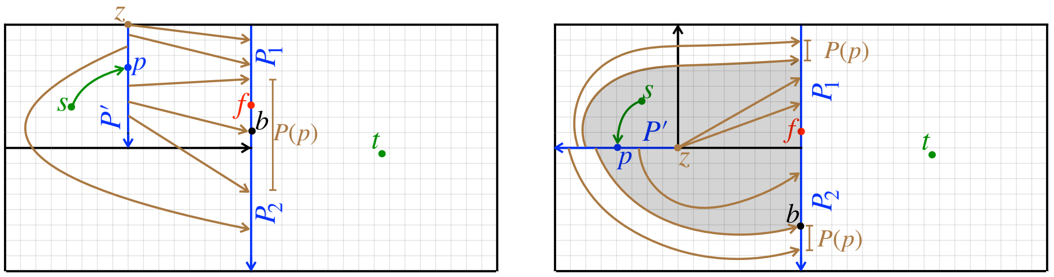


Fig. 5. When $f \in Q$: The vertical (blue) separator path P (in this example the separator Q consists of just a single path P) is partitioned by f into P_1 and P_2. The vertex z is the first vertex of p's (blue) path P'. The brown paths are the ways that z can reach P in $\hat{H}^\circ \setminus Q$. The vertex b is the first vertex of P_2 that is reachable from p in $\hat{H}^\circ \setminus Q$. In the left image, f lies inside the interval of vertices $P(p)$ that are reachable from p (b is therefore stored in f). In the right image, f lies outside the two intervals $P(p)$ (and b is therefore stored in s). The shaded area is the cycle C in the proof of Claim 3.1.

Q). Moreover, since anything reachable in $\hat{H}^\circ \setminus Q$ from p is also reachable from z, we have that $P(p) \subseteq P(z)$. In fact, planarity dictates the following:

Claim 3.1. *The set $P(p)$ consists of at most two intervals of consecutive vertices of $P(z)$.*

Proof. Consider two vertices $u, v \in P(p)$ that belong to two disjoint intervals of $P(z)$. Let C be the (undirected) cycle formed by the p-to-v path, the p-to-u path, and $P[u, v]$ (see Figure 5). The vertex z must be enclosed by C, otherwise, the path from z to any vertex of $P[u, v]$ must intersect the p-to-v path or the p-to-u path (and is thus reachable from p as well in contradiction to the two intervals being disjoint). Since z is enclosed by C, any vertex of P that is reachable from z and does not belong to $P[u, v]$ is also reachable from p (since the path to it from z must intersect the p-to-v path or the p-to-u path). □

For every $s \in G$, for every pair of ancestor pieces $H', \hat{H}$ of s, for every two paths $P' \in \partial H'$, and P of the separator Q of $\hat{H}$, s stores the identities of the endpoints of the two intervals of $P(p)$ in $P(s)$ (where p and z are as defined above). For every $f \in G$, for every pair of ancestor pieces $H', \hat{H}$ of f, for every two paths P' of the boundary of the sibling of H', and P of the separator Q of $\hat{H}$, f stores the identity of the first vertex v of $P(z)$ that is after f on P' (where z is the first vertex of P').

We can finally describe how to find $b_2 = \text{first}_{\hat{H}^\circ \setminus Q}(s, P_2)$: If the vertex v stored in the label of f falls inside one of the two intervals stored in the label of s (for the common pair P', P) then $b_2 = v$ and we have it stored in f. Otherwise, b_2 is the earliest starting endpoint that is later than f among the two endpoints of the two intervals stored in the label of s for P' and P (if both intervals are before f on P then b_2 does not exist).

We have therefore achieved our first goal: we found $b_1 = \text{first}_{\hat{H}^\circ \setminus Q}(s, P_1)$ and $b_2 = \text{first}_{\hat{H}^\circ \setminus Q}(s, P_2)$. It remains to check if there is a replacement path $R[b, t]$ in $G \setminus \{f\}$ from some $b \in \{b_1, b_2\}$. We can assume that such a path starts from b, continues to $q = \text{first}_{G \setminus \{f\}}(b, P_1)$ or to $q = \text{first}_{G \setminus \{f\}}(b, P_2)$, continues from q to $\text{last}_{G \setminus \{f\}}(t, P_1)$ (if $q \in P_1$) or to $\text{last}_{G \setminus \{f\}}(t, P_2)$ (if $q \in P_2$), and then finally continues to t. See Figure 4 (left). In Section 4 we present a labeling scheme that labels the vertices of P s.t. given the labels of any $b, f \in P$ we can find both $\text{first}_{G \setminus \{f\}}(b, P_1)$ and $\text{first}_{G \setminus \{f\}}(b, P_2)$. A symmetric labeling finds $\text{last}_{G \setminus \{f\}}(t, P_1)$ and $\text{last}_{G \setminus \{f\}}(t, P_2)$. We shall refer to this labeling as the *secondary* labeling. The secondary labels are stored in the labels of s and f along with the identities of the endpoints of the intervals (for s) and the first vertex v (for f) as described above. This way, the secondary label of b_1 is available in the label of s and the secondary label of b_2 is available in the label of f (if $b_2 = v$ falls inside one of the two intervals stored in s) or of s (if b_2 does not fall inside one of the two intervals). Recall that if s or f is an apex then we stored b_2 explicitly (in either s or f), in this case we additionally store the secondary label of b_2.

Before moving on to describe the secondary labeling scheme, there is one delicate issue: This labeling scheme requires that the two endpoints of the path P lie on the same face. Recall that we are working under the assumption that the replacement path R does not touch any path of the fundamental cycle separator used to separate $\hat{H}$ other than P. Hence, before computing the secondary labels we make an incision along the edges of the cycle separator that do not belong to P. Under our assumption the incision does not affect the replacement path, and now the endpoints of P indeed lie on a single face of $\hat{H}$.

4 WHEN THE QUERY VERTICES LIE ON A KNOWN PATH

In this section we present the secondary labeling scheme (with polylogarithmic-size labels), which addresses the following problem: We are given a directed planar graph G and a single path P in G whose endpoints lie on the same face. We need to label the vertices of P such that given the labels of any two vertices b, f of P we can find the first vertex $p < f$ of P that is reachable from b in $G \setminus \{f\}$ and the first vertex $p > f$ of P that is reachable from b in $G \setminus \{f\}$.

4.1 An auxiliary procedure

We begin with an auxiliary procedure that will be useful for our labeling. In this procedure, we wish to label the vertices of P, such that given the labels of any two vertices b, f such that f is before b on P (i.e., $f < b$), we can find the first vertex $p > f$ of P that is reachable in G from b using a path that does not touch f or any vertex before f (i.e., does not touch any vertex $v \leq f$ of P).

Let H_P be the graph composed of the path P and the following additional edges: for every pair of vertices $u, v \in P$ where $u > v$, we add an edge (u, v) iff (1) there exists a u-to-v path in G that does not touch P before v, and (2) there is no such w-to-v path for any $w > u$. The following claim shows that instead of working with G, we can work with H_P:

Claim 4.1. *Given any $f < b$, the first vertex $p > f$ that is reachable from b using a path that does not touch f or any vertex of P before f, is the same in G and in H_P.*

Proof. Let S be the corresponding b-to-p path in G for $f < p < b$. The path S visits no vertex of P before f (by definition of S) and no vertex of P between f and p (by definition of p). Hence, in H_P there is an edge (u, p) for some u where $u = b$ or $u > b$, and so S is represented in H_P (by a path composed of a b-to-u prefix along P followed by the single edge (u, p)). In the other direction, let R be the corresponding b-to-p path in H_P. The path R visits no vertex of P before f (by definition of R) and every edge (u, v) of R corresponds to a path in G that visits no vertex of P before v (and therefore visits no vertex of P before f), hence R is appropriately represented in G. □

We call the edges of H_P that are not edges of P *detours*. We will use the fact that detours do not cross (i.e., they form a laminar family):

Claim 4.2. *If there is a detour (u, v) then there is no detour (w, x) with $v < x < u < w$.*

Proof. By concatenating the (w, x) detour, the x-to-u subpath of P, and the (u, v) detour, we get a path in G that does not touch P before v but starts at a vertex $w > u$ contradicting condition (2) of the H_P edges. □

We now explain what needs to be stored to facilitate the auxiliary procedure. We define the *size* $|d|$ of a detour $d = (u, v)$ as the number of vertices of P between v and u. We say that a vertex of P is *contained* in a detour $d = (u, v)$ if it lies on P between v and u. We say that a detour d' is *contained* in detour d if all vertices that are contained in d' are also contained in d. Notice that, from Claim 4.1, the vertex p sought by the auxiliary procedure is an endpoint of the largest detour d that contains b and does not contain f. To find d, every vertex v of P stores a set of $O(\log n)$ nested detours $d_1^v, d_2^v, \ldots$ where d_1^v is the largest detour containing v, and d_{i+1}^v is the largest detour of size $|d_{i+1}^v| \le |d_i^v|/2$ containing v. By Claim 4.2, this set of nested detours is well defined. Additionally, for each d_i^v, v stores the largest detour $\hat{d}_i^v$ that is strictly contained in d_i^v but does not contain d_{i+1}^v. See Figure 6.

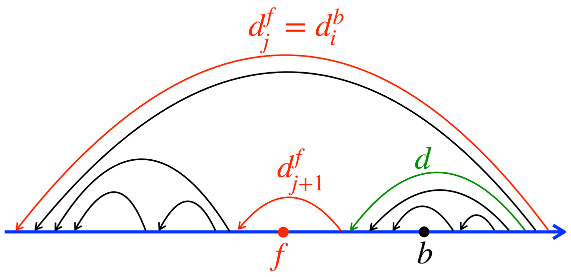


Fig. 6. A nested system of $O(\log n)$ detours $d_1^f, d_2^f, \ldots$ of f (in red) used in the auxiliary procedure. The (green) detour d is the largest detour that contains b and does not contain f. Either $d = \hat{d}_j^f$ and is stored in f, or $d = d_{i+1}^b$ and is stored in b.

Consider the smallest detour $d_j^f = d_i^b$ that is saved by both b and f. If such a detour does not exist then there is no detour that contains both b and f and so $d = d_1^b$ is stored in b. If $|d| > |d_j^f|/2$ then d must be the largest detour that is contained in d_j^f but does not contain d_{j+1}^f. In other words, $d = \hat{d}_j^f$ and is stored in f. If on the other hand $|d| \le |d_j^f|/2$, then $|d| \le |d_i^b|/2$ and by the choice of d_i^b we have that d_{i+1}^b does not contain f and so $d = d_{i+1}^b$ and is stored in b. This completes the description of the auxiliary procedure.

4.2 The labeling

Equipped with the above auxiliary procedure, we are now ready to describe the secondary labeling scheme. Recall that there are four cases to consider: Given $b, f \in P$ where b can be before/after f we wish to find the first vertex p before/after f that is reachable from b in $G \setminus \{f\}$. There are four cases to consider:

Given $f < b$, find the first vertex $p < f$ that is reachable from b in $G \setminus \{f\}$. Consider the b-to-p path R in $G \setminus \{f\}$. Let r be the first vertex of R that belongs to P and $r < f$. Let r' be the vertex of P that precedes r on R. Note that $r < f$, that $r' > f$, and that the r'-to-r subpath of R (which we call a *bypass* of f and denote $R[r', r]$) is internally disjoint from P and it either emanates to the left or to the right of P. Moreover, since the endpoints of P lie on the same face, then if R emanates at r' to the left (resp. right) of P it must enter P at r from the left (resp. right) of P. This means that we can assume w.l.o.g that the bypass $R[r', r]$ is the largest such bypass (in terms of the number of vertices of P between r and r'). This is because any smaller bypass that is on the same side of P as $R[r', r]$ is either contained in $R[r', r]$ (in which case we might as well use $R[r', r]$) or intersects $R[r', r]$ implying (in contradiction) that there is a larger bypass. See Figure 7.

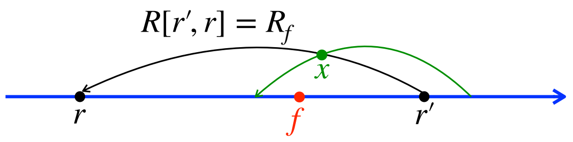


Fig. 7. A bypass $R[r', r]$ (in black) of f. If there was an intersecting bypass (in green) that is larger than $R[r', r]$ then the green subpath before x and the black subpath after x would constitute a larger bypass.

We let each vertex $f \in P$ store the endpoints of the largest bypass L_f (resp. R_f) that is internally disjoint from P, emanates left (resp. right) of P at a vertex that is after f, and enters P from the left (resp. right) at a vertex that is before f. The vertex $f \in P$ also stores the first vertex L_f^+ (resp. R_f^+) of P that is before f and is reachable in $G \setminus \{f\}$ from the endpoint of L_f (resp. R_f) that is after f.

In order to find p, we first use the auxiliary procedure of Section 4.1 to find the first vertex $p' > f$ that is reachable in G from b using a path that does not touch any vertex of P before f. Then, we check whether p' is contained in L_f and if so we consider L_f^+ as a candidate for p. Similarly, we check whether p' is contained in R_f and if so we consider R_f^+ as a candidate for p. Finally, we return the earlier of the (at most two) candidates.

Given $f < b$, find the first vertex $p > f$ that is reachable from b in $G \setminus \{f\}$. This case is very similar to the previous case. The only differences are: (1) we let every vertex $f \in P$ store the first vertex L_f^- (resp. R_f^-) of P that is after f and is reachable in $G \setminus \{f\}$ from the endpoints of L_f (resp. R_f), and (2) we add p' itself as a third possible candidate for p.

Given $b < f$, find the first vertex $p > f$ that is reachable from b in $G \setminus \{f\}$. This case and the next one are handled by small (but not symmetric) modifications of the previous two cases. Consider the b-to-p path R in $G \setminus \{f\}$. Let r be the first vertex of R that belongs to P and $r > f$. Let r' be the vertex of P that precedes r on R. The subpath $R[r', r]$ (which we call a *byway* of f) is internally disjoint from P, and if it emanates at r' to the left (resp. right) of P then it must enter r from the left (resp. right) of P. We can assume w.l.o.g that $R[r', r]$ is the smallest such byway (because any larger byway that is on the same side of P as $R[r', r]$ either contains $R[r', r]$ (in which case we might as well use $R[r', r]$) or intersects $R[r', r]$ implying (in contradiction) that there is a smaller byway. See Figure 8.

We let each vertex $f \in P$ store the endpoints of the smallest byway L_f (resp. R_f) containing f that is to the left (resp. right) of P. The vertex $f \in P$ also stores the first vertex L_f^- (resp. R_f^-) of P that is after f and is reachable in $G \setminus \{f\}$ from the endpoint of L_f (resp. R_f) that is after f.

In order to find p, we begin by finding the first vertex $p' < f$ that is reachable in G from b using a path that does not touch any vertex of P after f. This is done by using a variant of the auxiliary procedure of Section 4.1 with the following modification. In H_P we add the detour (u, v) iff (1) there exists a u-to-v path in G that does not touch P before v, and (2) there is no such u-to-w path for any $w < v$. After finding p' using this mechanism, we check whether p' appears on P before the starting point of the byway L_f, and if so we consider L_f^- as a candidate for p. Similarly, we check whether p' appears on P before the starting point of the byway R_f and if so we consider R_f^- as a candidate for p. Finally, we return the earlier of the (at most two) candidates.

Given $b < f$, find the first vertex $p < f$ that is reachable from b in $G \setminus \{f\}$. This case is very similar to the previous case. The only differences are: (1) we let every vertex $f \in P$ store the first

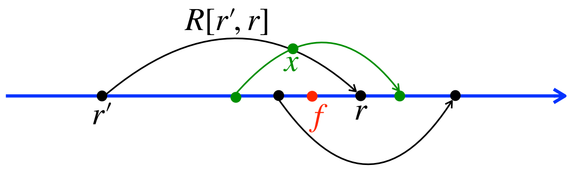


Fig. 8. Two byways of f (in black), the top one is $R[r', r]$. If there was an intersecting byway (in green) that is smaller than $R[r', r]$ then the green subpath before x and the black subpath after x would constitute a smaller byway.

vertex L_f^+ (resp. R_f^+) of P that is before f and is reachable in $G \setminus \{f\}$ from the endpoints of L_f (resp. R_f), and (2) we add p' itself as a third possible candidate for p.

REFERENCES

[1] I. Abraham, S. Chechik, and C. Gavoille. Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels. In *44th STOC*, pages 1199–1218, 2012.
[2] I. Abraham, S. Chechik, C. Gavoille, and D. Peleg. Forbidden-set distance labels for graphs of bounded doubling dimension. *ACM Trans. Algorithms*, 12(2):22:1–22:17, 2016.
[3] N. Alon and R. Nenadov. Optimal induced universal graphs for bounded-degree graphs. In *28th SODA*, pages 1149–1157, 2017.
[4] S. Alstrup, S. Dahlgaard, and M. B. T. Knudsen. Optimal induced universal graphs and adjacency labeling for trees. *J. ACM*, 64(4):27:1–27:22, 2017.
[5] S. Alstrup, H. Kaplan, M. Thorup, and U. Zwick. Adjacency labeling schemes and induced-universal graphs. *SIAM J. Discret. Math.*, 33(1):116–137, 2019.
[6] A. Bar-Natan, P. Charalampopoulos, P. Gawrychowski, S. Mozes, and O. Weimann. Fault-tolerant distance labeling for planar graphs. *Theor. Comput. Sci.*, 918:48–59, 2022.
[7] S. Baswana, K. Choudhary, and L. Roditty. Fault-tolerant subgraph for single-source reachability: General and optimal. *SIAM J. Comput.*, 47(1):80–95, 2018.
[8] S. Baswana, U. Lath, and A. S. Mehta. Single source distance oracle for planar digraphs avoiding a failed node or link. In *23rd SODA*, pages 223–232, 2012.
[9] N. Bonichon, C. Gavoille, and A. Labourel. Short labels by traversal and jumping. *Electronic Notes in Discrete Mathematics*, 28:153–160, 2007.
[10] P. Charalampopoulos, P. Gawrychowski, Y. Long, S. Mozes, S. Pettie, O. Weimann, and C. Wulff-Nilsen. Almost optimal exact distance oracles for planar graphs. *J. ACM*, 70(2):12:1–12:50, 2023.
[11] P. Charalampopoulos, S. Mozes, and B. Tebeka. Exact distance oracles for planar graphs with failing vertices. In *30th SODA*, pages 2110–2123, 2019.
[12] K. Choudhary. An optimal dual fault tolerant reachability oracle. In *43rd ICALP*, pages 130:1–130:13, 2016.
[13] B. Courcelle, C. Gavoille, and M. M. Kanté. Compact labelings for efficient first-order model-checking. *Journal of Combinatorial Optimization*, 21(1):19–46, 2009.
[14] B. Courcelle and A. Twigg. Constrained-path labellings on graphs of bounded clique-width. *Theory of Computing Systems*, 47(2):531–567, 2010.
[15] J. Feigenbaum, D. R. Karger, V. S. Mirrokni, and R. Sami. Subjective-cost policy routing. *Theor. Comput. Sci.*, 378(2):175–189, 2007.
[16] C. Gavoille, M. Katz, N. A. Katz, C. Paul, and D. Peleg. Approximate distance labeling schemes. In *9th ESA*, pages 476–487, 2001.
[17] C. Gavoille, D. Peleg, S. Pérennes, and R. Raz. Distance labeling in graphs. *Journal of Algorithms*, 53(1):85–112, 2004.
[18] L. Georgiadis, D. Graf, G. F. Italiano, N. Parotsidis, and P. Uznanski. All-pairs 2-reachability in $O(n^\omega \log n)$ time. In *44th ICALP*, pages 74:1–74:14, 2017.
[19] L. Georgiadis, G. F. Italiano, and N. Parotsidis. Strong connectivity in directed graphs under failures, with applications. In *28th SODA*, pages 1880–1899, 2017.
[20] M. Henzinger, A. Lincoln, S. Neumann, and V. V. Williams. Conditional hardness for sensitivity problems. In *8th ITCS*, pages 26:1–26:31, 2017.

[21] J. Holm, E. Rotenberg, and M. Thorup. Planar reachability in linear space and constant time. In *56th FOCS*, pages 370–389, 2015.

[22] T. Hsu and H. Lu. An optimal labeling for node connectivity. In *20th ISAAC*, 2009.

[23] G. F. Italiano, A. Karczmarz, and N. Parotsidis. Planar reachability under single vertex or edge failures. In *32nd SODA*, pages 2739–2758, 2021.

[24] S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. *SIAM Journal on Discrete Mathematics*, 5(4):596–603, 1992.

[25] M. Katz, N. A. Katz, A. Korman, and D. Peleg. Labeling schemes for flow and connectivity. *SIAM J. Comput.*, 34(1):23–40, 2004.

[26] V. King and G. Sagert. A fully dynamic algorithm for maintaining the transitive closure. In *31st STOC*, pages 492–498, 1999.

[27] P. Klein and S. Mozes. Optimization algorithms for planar graphs. http://planarity.org. Book draft.

[28] A. Korman. Labeling schemes for vertex connectivity. *ACM Trans. Algorithms*, 6(2):39:1–39:10, 2010.

[29] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36(2):177–189, 1979.

[30] D. Peleg. Informative labeling schemes for graphs. *Theor. Comput. Sci.*, 340(3):577–593, 2005.

[31] C. Petersen, N. Rotbart, J. G. Simonsen, and C. Wulff-Nilsen. Near-optimal adjacency labeling scheme for power-law graphs. In *43rd ICALP*, pages 133:1–133:15, 2016.

[32] N. G. Rotbart. *New Ideas on Labeling Schemes*. PhD thesis, University of Copenhagen, 2016.

[33] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, 2004.

[34] A. D. Twigg. Compact forbidden-set routing. Technical Report UCAM-CL-TR-678, University of Cambridge, Computer Laboratory, Dec. 2006.

[35] J. van den Brand and T. Saranurak. Sensitive distance and reachability oracles for large batch updates. In *60th FOCS*, pages 424–435, 2019.