Fault-Tolerant Distance Labeling for Planar Graphs

Aviv Bar-Natan[a], Panagiotis Charalampopoulos[b], Paweł Gawrychowski[c], Shay Mozes[b], Oren Weimann[a]

[a]*University of Haifa, Israel*
[b]*Reichman University, Israel*
[c]*University of Wrocław, Poland*

Abstract

In fault-tolerant distance labeling we wish to assign short labels to the vertices of a graph G such that from the labels of any three vertices u, v, f we can infer the u-to-v distance in the graph $G \setminus \{f\}$. We show that any directed weighted planar graph (and in fact any graph in a graph family with $O(\sqrt{n})$-size separators, such as minor-free graphs) admits fault-tolerant distance labels of size $O(n^{2/3})$. We extend these labels in a way that allows us to also count the number of shortest paths, and provide additional upper and lower bounds for labels and oracles for counting shortest paths.

Keywords: forbidden-set distance labels, planar graphs, fault-tolerant distance labels, counting shortest paths

1. Introduction

Computing distances in graphs is one of the most basic and important problems in graphs theory, both from theoretical and practical points of view. In this work we consider distance labeling schemes, in which one preprocesses a network to assign labels to the vertices, so that the distance between any two vertices u and v can be recovered from just the labels of u and v (and no other information). The main criteria of interest are foremost the size of the label, and to a lesser extent the time it takes to recover the distance from a given pair of labels (query time). Distance labeling schemes are useful in the distributed setting, where it is advantageous to be able to infer distances based only on local information such as the labels of the source and destination. This is the case

in communication networks or in disaster stricken areas, where communication with a centralized entity is infeasible or downright impossible.

Considering the latter scenario of disaster management, it is not only likely that a disastrous event makes communication with a centralized entity impossible, but also that parts of the network are affected by the disaster, and that only shortest paths that avoid affected parts of the network should be considered when computing distances. Forbidden-set distance labeling schemes assign labels to vertices, so that, for any pair of vertices u and v, and any set F of failed vertices, the length of a shortest u-to-v path that avoids all vertices in F can be recovered just from the labels of u, v, and of the vertices in F. In this work we study forbidden-set distance labeling schemes in directed planar networks. We also study the extension of such schemes to capture not only the distance from u to v, but also the number of distinct u-to-v shortest paths.

For unweighted graphs, we measure the label size in bits. For weighted graphs and queries concerning lengths of the shortest paths, we assume that the distance between any two vertices fits in a single machine word, and measure the label size in words. For queries concerning the number of shortest paths, unless mentioned otherwise, we assume that the number of shortest paths between any two vertices fits in a single machine word, and measure the label size in words.

1.1. Related work

Labeling schemes provide a clean and natural model for studying how to distribute information about a graph. Problems considered in this model include adjacency [1, 2, 3, 4, 5, 6], flows and connectivity [7, 8, 9], and Steiner tree [10]. See [11] for a recent survey. We specifically focus on distance labeling schemes.

Distance labeling schemes. Embedding distance information into labels was studied by Graham and Pollak [12] in the 1970's in the so-called squashed cube model. In 2000, Peleg [13] formalized the notion of distance labeling schemes, and provided schemes with polylogarithmic label size (in bits) and query time for trees, interval graphs and permutation graphs. Gavoille et al. [14] showed that for general graphs, the label size is $\Theta(n)$, and for trees, $\Theta(\log^2 n)$. For (unit-weight) planar graphs they showed a lower bound of $\Omega(n^{1/3})$, and an upper bound of $O(\sqrt{n} \log n)$ bits. The $O(\sqrt{n} \log n)$ upper bound is simple (just store the distance to each of the $O(\sqrt{n})$ vertices of a planar separator, and then recurse on both sides of the separator), and was recently improved to $O(\sqrt{n})$ [15]. However, the rare *polynomial* gap between the lower and upper bound remains an interesting and important

2

open problem. For weighted planar graphs Gavoille et al. gave tight (up to polylogarithmic factors) $\tilde{\Theta}(n^{1/2})$ upper and lower bounds.[1]

Approximate distance labeling schemes. Since exact distance labels typically require polynomial size labels [14], researchers have sought smaller labels that yield approximate distances. Gavoille et al. [16] studied such labels for general graphs and various graph families. Specifically, for planar graphs, they presented $O(n^{1/3} \log n)$-bit labels that provide a 3-approximation of the distance. In the same year, Gupta et al. [17] presented smaller 3-approximate labels, requiring only $O(\log^2 n)$ bits, and Thorup gave $(1 + \epsilon)$-approximate labels of size $O(\log n/\epsilon)$, for any fixed $\epsilon > 0$ [18]. The latter result was generalized to H-minor free graphs by Abraham and Gavoille in [19].

Forbidden-set distance labeling schemes. Forbidden-set labels were introduced in the context of routing labels by Feigenbaum et al. [20, 21], and studied by several others [22, 23, 24, 25, 26]. Exact forbidden-set labeling schemes of polylogarithmic size are given in [22, 24] for graphs of bounded treewidth or cliquewidth. For unweighted graphs of bounded doubling dimension, forbidden-set labels with polylogarithmic size and $(1 + \epsilon)$-stretch are also known [25]. For undirected planar graphs, and for any fixed $\epsilon > 0$, Abraham et al. [26] presented a forbidden-set labeling scheme of polylogarithmic size such that a $(1 + \epsilon)$-approximation of the shortest path between vertices u and v that avoids a set F of failed vertices can be recovered from the labels of u, v, and the labels of the failed vertices in $\tilde{O}(|F|^2)$ time.

Other related work on planar graphs. There are many other concepts related to distances in the presence of failures. In the *replacement paths* problem, we are given a graph along with a source and sink vertices, and the goal is to efficiently compute all shortest paths between the source and the destinations for every possible single-edge failure in the graph. In planar graphs, this problem can be solved in nearly linear time [27, 28, 29]. Moreover, for the single-source single-failure version of the problem (i.e. when only the source vertex is fixed at construction time, and the query specifies just the target and a single failed vertex), Baswana et al. [30] presented an oracle with size and construction time $O(n \log^4 n)$ that answers queries in $O(\log^3 n)$ time. Building upon this oracle, they then present an oracle of size $\tilde{O}(n^2/q)$ supporting arbitrary distance queries subject to a single failure in time $\tilde{O}(q)$ for any $q \in [1, n^{1/2}]$. The authors of [31] show how to construct in $\tilde{O}(n)$ time an

---

[1] The $\tilde{O}(\cdot)$ notation suppresses $\log^{O(1)} n$ factors.

oracle of size $\tilde{O}(n)$ that, given a source vertex u, a target vertex v, and a set F of k faulty vertices, reports the length of a shortest u-to-v path in $G \setminus F$ in $\tilde{O}(\sqrt{kn})$ time. They further show that for any $r \in [1, n]$ there exists an $\tilde{O}(\frac{n^{k+1}}{r^{k+1}}\sqrt{nr})$-size oracle that answers queries in time $\tilde{O}(k\sqrt{r})$. Recently, Italiano et al. [32] gave an oracle of size $O(n \log n)$ and construction time $O(n \log^2 n / \log \log n)$ that supports reachability queries subject to a single failure in time $O(\log n)$.

Another related concept is that of dynamic distance oracles. Here a graph is preprocessed so as to efficiently support distance queries between arbitrary pairs of vertices as well as updates to the graph. Updates may include deletion of edges or vertices (decremental updates), or also addition of new edges and vertices (fully dynamic). Fakcharoenphol and Rao [33] presented distance oracles for planar graphs that require $\tilde{O}(n^{2/3})$ and $\tilde{O}(n^{4/5})$ amortized time per update and query for non-negative and arbitrary edge-weight updates respectively.[2] The space required by these oracles is $O(n \log n)$. The extensions of this result in [34, 35, 36, 31] yield a dynamic oracle for planar graphs that can handle arbitrary edge weight updates, edge deletions and insertions (not violating the planarity of the embedding) and vertex deletions, as well as answer distance queries, in $\tilde{O}(n^{2/3})$ time each.

Counting shortest paths in planar graphs. In the (non-faulty) counting version of shortest paths labeling, given the labels of vertices s and t we wish to return the *number* of shortest s-to-t paths in G (i.e. paths whose length is equal to $d(s, t)$). This problem (without faults) in planar graphs was recently studied in [37] where labels[3] of size $\Theta(\sqrt{n})$ were constructed under the assumption that the number of shortest paths between any two vertices fits in a constant number of machine words. In the general case where the numbers consist of L bits, the obtained labels consist of $O(\sqrt{n} \cdot L)$ bits. As already observed in [37], it is easy to construct an unweighted planar graph where $L = n - 1$ making the labels consist of $\Theta(n^{1.5})$ bits, that is, more than in a naive encoding storing the whole graph in every label. However, the following simple construction shows that we cannot hope to construct labels consisting of $o(n)$ bits without bounding L:

Fact 1. *There is no labeling scheme for counting shortest paths in planar graphs with labels consisting*

---

[2]Though this is not mentioned in [33], the query time can be made worst-case rather than amortized by standard techniques.

[3]In [37], the authors actually considered the oracle version of the problem, but their solution can be easily applied for labeling as well.

4

of o(n) bits.

Proof. Given n bits $b_0, \ldots, b_{n-1}$ we construct a graph consisting of a path $s = u_0 - u_1 - \cdots - u_{n-1}$ and another path $v_1 - v_2 - \cdots - v_n = t$ in which every edge is duplicated (i.e. there are two parallel edges between each pair v_i, v_{i+1}). Finally, for every $i = 0, \ldots, n-1$ such that $b_i = 1$, we add an edge $u_i - v_{i+1}$. Then the number of shortest s-to-t paths is exactly $\sum_{i=0}^{n-1} b_i \cdot 2^{n-1-i}$, and so by an encoding argument the total number of bits in the labels of s and t must be at least n. $\qquad\square$

Therefore, when counting shortest paths we will measure the size of a label in the number of machine words, each long enough to store the number of shortest paths between any two vertices in the graph.

We highlight one interesting application where our scheme for counting shortest s-to-t paths that avoid vertices $v_1, v_2, \ldots, v_k$ can be modified to obtain a better bound on the sizes of the labels in bits. Say that instead of counting such shortest paths we would like to check if avoiding vertices $v_1, v_2, \ldots, v_k$ increases the length of the shortest path. In such case, we only need to check if the number of shortest s-to-t paths that avoid vertices $v_1, v_2, \ldots, v_k$ is nonzero. Because the number of shortest paths is always at most 2^n, by well known properties of prime numbers, choosing a random prime p consisting of $\Theta(k \cdot \log n)$ bits guarantees that with high probability, for every $s, t, v_1, v_2, \ldots, v_k$, the number of shortest paths counted modulo p is nonzero if and only if the number of shortest paths is nonzero. Our scheme (as well as the scheme of [37]) can be used for counting modulo p, so we obtain labels consisting of $\tilde{O}(\sqrt{n} \cdot k)$ bits for such queries.

1.2. Our results

- In Section 3 we present a single-fault distance labeling scheme (forbidden-set labeling scheme for a set of cardinality 1) with label size $O(n^{2/3})$. By increasing the label-size by polylogarithmic factors, we show how to support queries in $\tilde{O}(\sqrt{n})$ time, and how to construct all labels in $\tilde{O}(n^{5/3})$ time. Our labeling scheme extends (with no overhead in the label size) to a labeling scheme for counting shortest paths (with a single fault).

- In Section 4 we extend the counting labels of [37] to the following fault-tolerant variant. Given the labels of vertices $s, t, v_1, v_2, \ldots, v_k$, we wish to return the number of s-to-t paths that avoid vertices $v_1, \ldots, v_k$ and whose length is equal to $d(s, t)$ (the original s-to-t distance in G). We show that the labeling of [37] (with labels of size $\tilde{O}(\sqrt{n})$) actually works in this more general

5

setting. A naive query to such labeling takes $\tilde{O}(\sqrt{n} \cdot k^2)$ time, we show how to improve this to $\tilde{O}(\sqrt{n} \cdot k)$.

- In Section 5 we show a lower bound of $\Omega(\sqrt{nL})$ on the label-size (in bits) for counting shortest paths (without faults), in graphs in which the number of distinct shortest paths between any two vertices consists of at most L bits.

- In Section 6 we show a lower bound on dynamic oracles for counting shortest paths, conditioned on the hardness of online boolean matrix-vector multiplication. We prove that for any dynamic shortest paths counting oracle in undirected planar graphs, either the query time or the update time must be $\Omega(\sqrt{n})$ (up to subpolynomial factors).

We focus on planar graphs but in fact all our results (except for the efficient preprocessing time and query time in Section 3) hold for any graph family with $O(\sqrt{n})$-size separators (such as H-minor free graphs and bounded genus graphs). This is also the case for the standard (i.e. without failures) labeling scheme of Gavoille et al. [14]. However, while their $\tilde{O}(n^{1/2})$-size labels are obtained with a straightforward application of separators, our $O(n^{2/3})$-size (fault-tolerant) labels are obtained with a non-standard and intricate use of separators.

2. Preliminaries

Throughout the paper we consider as input a weighted directed planar graph G, embedded in the plane. We assume that the input graph has no negative length cycles. We can transform the graph in a standard way, in $O(n \frac{\log^2 n}{\log \log n})$ time, so that all edge weights are non-negative and distances are preserved [38].

Separators and recursive decompositions. Miller [39] showed how to compute a Jordan curve that intersects the graph at a set of vertices $Sep(G)$ of size $O(\sqrt{n})$ and separates G into two pieces with at most $2n/3$ vertices each. Jordan curve separators can be used to recursively separate a planar graph until pieces have constant size. The authors of [40] show how to obtain a complete recursive decomposition tree $\mathcal{T}$ of G in $O(n)$ time. $\mathcal{T}$ is a binary tree whose nodes correspond to subgraphs of G (called *pieces*), with the root being all of G and the leaves being pieces of constant size. We identify each piece P with the node representing it in $\mathcal{T}$ (we can thus abuse notation and write $P \in \mathcal{T}$), with its boundary ∂P (i.e. vertices that belong to some separator along the recursive

6

decomposition used to obtain P), and with its separator $Sep(P)$. We denote by $\mathcal{T}[P, Q]$ the P-to-Q path in $\mathcal{T}$ (and also use $\mathcal{T}(P, Q]$, $\mathcal{T}[P, Q)$, and $\mathcal{T}(P, Q)$).

An r-*division* [41] of a planar graph, for $r \in [1, n]$, is a decomposition of the graph into $O(n/r)$ pieces, each of size $O(r)$, such that each piece P has $O(\sqrt{r})$ boundary vertices (denoted ∂P). Another desired property of an r-division is that the boundary vertices lie on a constant number of faces (called holes) of the piece. For every r larger than some constant, an r-division with few holes is represented in the decomposition tree $\mathcal{T}$ of [40]. It is convenient to describe the r-division by truncating $\mathcal{T}$ at pieces of size $O(r)$, that also satisfy the other required properties. We refer to those pieces (the leaves of $\mathcal{T}$ after truncation) as *regions* and denote by R_u the region containing vertex u (if u belongs to multiple regions, we arbitrarily designate one of them as R_u).

Dense distance graphs and FR-Dijkstra. The *dense distance graph* of a set of vertices U that lie on a constant number of faces of a planar graph H, denoted $DDG_H(U)$ is a complete directed graph on the vertices of U. Each edge (u, v) has weight $d_H(u, v)$, equal to the length of the shortest u-to-v path in H. $DDG_H(U)$ can be computed in time $O((|U|^2 + |H|) \log |H|)$ using the multiple source shortest paths (MSSP) algorithm [34, 42]. Thus, computing $DDG_P(\partial P)$ over all pieces of the recursive decomposition of G requires time $O(n \log^2 n)$ and space $O(n \log n)$. We next give a –convenient for our purposes– interface for FR-Dijkstra [33], which is an efficient implementation of Dijkstra's algorithm on any union of DDGs. The algorithm exploits the fact that, due to planarity, certain submatrices of the adjacency matrix of $DDG_H(U)$ satisfy the Monge property. (A matrix M satisfies the Monge property if, for all $i < i'$ and $j < j'$, $M_{i,j} + M_{i',j'} \leq M_{i',j} + M_{i,j'}$ [43].) The interface is specified in the following theorem, which was essentially proved in [33], with some additional components and details from [36, 38].

Theorem 2 ([33, 36, 38]). *Given a set Y of DDGs, Dijkstra's algorithm can be run on the union of any subset of Y with $O(N)$ vertices in total (with multiplicities) and an arbitrary set of $O(N)$ extra edges in time $O(N \log^2 N)$.*

3. Single-Fault Labeling for Reporting Shortest Paths

Warm-up. As a warm-up, we first sketch a simple labeling scheme that assigns a label of size $O(n^{4/5})$ to each vertex. Consider an r-division for $r = n^{4/5}$, and let $\mathcal{R}$ be the set of its regions. The label of each vertex u consists of the following:

(a) The r-division $\mathcal{R}$. Space: $O(n/r)$.

(b) For each region R in the r-division, the length of the shortest path in G, among paths that are internally disjoint from R, from u to $\bigcup_{P \in \mathcal{R}} \partial P$, and from $\bigcup_{P \in \mathcal{R}} \partial P$ to u. There are $O(n/r)$ regions and for each of them we store $O(n/r \cdot \sqrt{r})$ distances. Space: $O(n^2/r^{3/2})$.

(c) The region R_u and the ∂R_u-to-∂R_u distances in $G \setminus \{u\}$. Space: $O(r)$.

The space is thus $O(n/r + n^2/r^{3/2} + r) = O(n^{4/5})$.

Let us now consider a query (u, v, f), and assume, for simplicity, that no two of u, v and f are contained in a single region. We have two cases. If there is a shortest u-to-v path in $G \setminus \{f\}$ that is vertex-disjoint from R_f, then the u-to-∂R_v distances among paths internally-disjoint from R_f (item (b)), together with R_v, which is stored for v (item (c)), allow us to retrieve the length of this path. In the other case, we employ the u-to-∂R_f distances among paths internally-disjoint from R_f (item (b)), the information stored in item (c) for f, and the ∂R_f-to-v distances among paths internally-disjoint from R_f (item (b)).

It is not difficult to combine this approach with the distance-labeling scheme of Gavoille et al. [14] for the failure-free setting to obtain labels of size $O(n^{3/4})$. (Item (b) has to be modified to store distances to separators of ancestors of R_u instead of distances to $\bigcup_{P \in \mathcal{R}} \partial P$, requiring $O(n^{3/2}/r)$ space.) In the approach that we present below, we rely on separators in a more sophisticated and delicate manner to obtain labels of size $O(n^{2/3})$.

The label. Recall that an r-division is represented by a decomposition tree $\mathcal{T}$, whose root corresponds to G. The internal nodes of $\mathcal{T}$ correspond to pieces of G. The two children of a piece $P \in \mathcal{T}$ are the subgraphs of P external and internal to $Sep(P)$. The leaves of $\mathcal{T}$ are the regions of the r-division.

The label of each vertex u in G consists of the following information:

(i) The entire decomposition tree $\mathcal{T}$. Space: $O(n/r)$.

(ii) For each region R in the r-division, the shortest path distances in G from u to ∂R among paths that are internally disjoint from R. There $O(n/r)$ regions and each of them has $O(\sqrt{r})$ boundary vertices. Space: $O(n/\sqrt{r})$.

(iii) The region R_u and the ∂R_u-to-∂R_u distances in $G \setminus \{u\}$. Space: $O(r)$.

(iv) For each piece $P \in \mathcal{T}$ with sibling Q, for each $p \in \partial P \setminus Q$, the shortest path distance from u to p in $G \setminus (P \cup Q) \cup \{p\}$, and the shortest path distance from p to u in $G \setminus Q$. Space: $O(\sum_{P \in \mathcal{T}} \partial P) = O(n/\sqrt{r})$, c.f. [44].

(v) For each ancestor piece P of R_u in $\mathcal{T}$, for each vertex p of $Sep(P) \setminus \partial P$, the shortest path distance from u to p among paths in $P \setminus \partial P$ that are internally disjoint from $Sep(P)$, and the shortest path distance in $P \setminus \partial P$ from p to u. Space: $O(\sqrt{n})$, c.f. [44].

The overall space required by the above five items is $O(n/r + n/\sqrt{r} + r + n/\sqrt{r} + \sqrt{n})$, which is $O(n^{2/3})$ for $r = n^{2/3}$.

The query. Upon query (u, v, f) we say that a path is a (u, v, f)-path if it is a u-to-v path in G that avoids f, and we seek the shortest (u, v, f)-path, which we denote by S. Let X denote the lowest node in $\mathcal{T}$ that is an ancestor of R_f and of at least one of $\{R_u, R_v\}$. Let us assume without loss of generality that X is an ancestor of R_u. We return the minimum of the following three:

1. S includes a vertex of ∂R_f.

   The length of this path is found with a Single-Source Shortest-Paths (SSSP) computation on the (non-planar) graph G_1 whose vertices are u, v, and $\partial R_f \setminus \{f\}$ and whose edges are in one-to-one correspondence with the distances specified below, i.e. for each a-to-b distance, there is an edge from a to b with length equal to that distance:

   - the u-to-$\partial R_f \setminus \{f\}$ distances from item (ii) in u's label (or the u-to-$\partial R_f \setminus \{f\}$ distances in $R_f \setminus \{f\}$, which can be computed from item (iii), if $R_u = R_f$);

   - the $\partial R_f \setminus \{f\}$-to-$\partial R_f \setminus \{f\}$ distances from item (iii) in f's label;

   - the $\partial R_f \setminus \{f\}$-to-$v$ distances from item (ii) in v's label (or the $\partial R_f \setminus \{f\}$-to-$v$ distances in $R_f \setminus \{f\}$, which can be computed from item (iii), if $R_v = R_f$).

2. S avoids R_f but includes a boundary vertex of some piece on $\mathcal{T}[X, R_f]$.

   The length of this path is found with a SSSP computation on the graph G_2 whose vertices are u, v, and ∂P of all nodes P that are siblings of some node Q on the X-to-R_f path in $\mathcal{T}$. The edges are in one-to-one correspondence with the u-to-∂P distances from item (iv) in u's label and the ∂P-to-v distances from item (iv) in v's label.

3. S avoids all boundary vertices of all the pieces on $\mathcal{T}[X, R_f]$.

This is required only for the case where the lowest common ancestor of R_u and R_v is not an ancestor of R_f (otherwise, it is an ancestor of X and a u-to-v path cannot avoid the boundary vertices of X). The length of this path is found with a SSSP computation on the graph G_3 whose vertices are u, v, and $Sep(P) \setminus \partial P$ of all nodes P on $\mathcal{T}(X, R_u)$. The edges are in one-to-one correspondence with the u-to-$Sep(P)$ distances from item (v) in u's label and the $Sep(P)$-to-v distances from item (v) in v's label. If $R_u = R_v$, the shortest path may not cross any of these separators; in that case the distance may be retrieved by a single SSSP computation in $R_u \setminus \{f\}$ (item (iii)).

Correctness. Let us consider the three options for the shortest (u, v, f)-path S (an illustration is provided in Figure 1).

1. S includes a vertex of ∂R_f. Let a (resp. b) denote the first (resp. last) vertex of S that belongs to $\partial R_f \setminus \{f\}$. The path S can be partitioned into a u-to-a prefix, an a-to-b infix, and a b-to-v suffix. All three subpaths are represented in G_1, and all paths represented in G_1 do not include f.

2. S avoids R_f but includes a boundary vertex of some piece on $\mathcal{T}[X, R_f]$. First observe that all u-to-v paths in G_2 avoid some (not necessarily proper) ancestor of R_f and therefore also avoid f. To see that S is represented in G_2, let Q denote the unique piece on $\mathcal{T}(X, R_f]$ such that S avoids Q but visits its sibling P (such a piece Q must exist because S avoids R_f but visits some piece on $\mathcal{T}[X, R_f]$). Since S visits P it must visit some vertex of ∂P. Let p be the first such vertex of S. Partition S into a shortest u-to-p path in $G \setminus (Q \cup P) \cup \{p\}$ and a shortest p-to-v path in $G \setminus Q$. These two subpaths are represented in G_2.

3. S avoids all boundary vertices of all the pieces on $\mathcal{T}[X, R_f]$. If S does not visit ∂R_u (and thus $R_u = R_v$) then we find S with an SSSP computation in $R_u \setminus \{f\}$. Otherwise, S visits a separator vertex of some piece that is a proper ancestor of R_u. Let P be the rootmost such piece. Since S avoids ∂X we have that S is restricted to X and hence P is a descendant of X. In fact, P must be a *proper* descendant of X (otherwise, S visits $Sep(X)$ and therefore visits the boundary of both child-pieces of X including the one on $\mathcal{T}[X, R_f]$, a contradiction). We therefore have that $P \in \mathcal{T}(X, R_u)$ and S is restricted to P. Also observe that S avoids
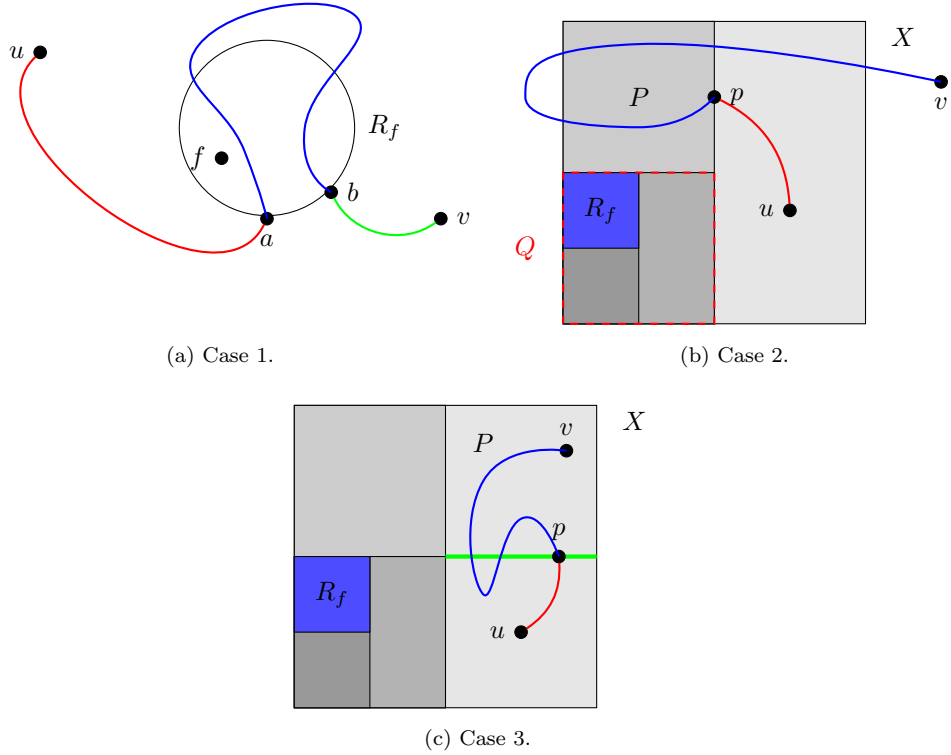
10

(a) Case 1.

(b) Case 2.

(c) Case 3.

Figure 1: An illustration of the three different cases that arise for the query. In the figures we assume that $u, v \notin R_f$ and the different colors in each path represent its decomposition as defined in the proof of correctness. In the figure for Case 2, the blue piece denotes R_f, while the siblings of its ancestors in $\mathcal{T}(X, R_f]$ are denoted by different scales of gray; the deeper the piece is in $\mathcal{T}$, the darker its color. Piece Q is denoted be the red-dashed rectangle. For Case 3, the setting is the same and in our illustration P is the child of X that is not an ancestor of R_f. $Sep(P)$ is denoted by green.

∂P because otherwise S must visit a separator vertex of some ancestor of P, contradicting P being rootmost. Let p be the first vertex of S that belongs to $Sep(P)$. S can be decomposed into a shortest path from u to p in $P \setminus \partial P$ that is internally disjoint from $Sep(P)$, and a suffix that is a shortest path from p to v in $P \setminus \partial P$; S is thus represented in G_3. To see that no path represented in G_3 contains f, observe that P may contain f, but since R_f is not a descendant of P, f must be a vertex of ∂P and so is not visited by any path represented in G_3.

We thus arrive at the following result.

Theorem 3. *Given a directed planar graph G of size n, with real edge-lengths, we can assign an*

$O(n^{2/3})$-size label to each vertex of G such that upon query (u, v, x), where $u, v, x \in V(G)$, the length of the shortest u-to-v path in $G \setminus \{x\}$ can be retrieved from the labels of u, v and x.

Remark. Any graph G of size n from a family of graphs that hereditarily admits $O(\sqrt{n})$-size separators (such as H-minor free graphs and bounded genus graphs) can be recursively decomposed so that we get an r-division (perhaps not with the few-holes property). As our labeling scheme does not require the few-holes property, Theorem 3 actually applies to any such graph family.

Extension for counting. We now show how to extend our single-fault labeling from reporting u-to-v shortest paths in $G \setminus \{f\}$ to counting the number of u-to-v shortest paths in $G \setminus \{f\}$. Our modification does not increase the label size (assuming that each number we store fits into a single word, see the discussion in the introduction). However, the efficient query algorithm cannot be applied, leading to $\tilde{O}(n^{2/3})$ query time.

In order to extend the labeling scheme for counting, for every u-to-v shortest path distance which is stored in our label, we also store the number of such u-to-v shortest paths. The change in query time is that instead of the SSSP computations on G_1, G_2, G_3 we use an SSSP computation that counts shortest paths. That is, for each edge in G_i there is a value representing its multiplicity (the value we added to the label), and we want to compute the number of shortest paths with respect to the multiplicities. This extension can be achieved by a trivial extension to Dijkstra's algorithm, resulting in $\tilde{O}(n^{2/3})$ query time (In contrast, FR-Dijkstra has no known extension for counting shortest paths). The following lemma proves the correctness of our labeling scheme.

Lemma 4. *Every shortest path from u to v in $G \setminus \{f\}$ is represented exactly once in the query graphs G_1, G_2, G_3.*

Proof. The same argument as in the correctness subparagraph proves that every shortest path is represented at least once in the query graphs. It remains to show that every path is represented at most once. Let us consider the three cases for a shortest (u, v, f)-path:

1. S includes a vertex of ∂R_f. S is not represented in G_2, G_3 because every path that is represented there must avoid an ancestor of R_f. S is represented exactly once in G_1 because it has a unique decomposition into subpaths $S_1 S_2 S_3$ where S_1 is from u to the first vertex b_1 of S in ∂R_f, S_2 is from b_1 to the last vertex b_2 of S in ∂R_f, and S_3 is from b_2 to v.

2. S avoids R_f but includes a boundary vertex of some piece on $\mathcal{T}[X, R_f]$. S is not represented in G_1 because all the paths that are represented there touch R_f, it is also not represented in G_3 since every path there avoids all boundary vertices of all pieces in $\mathcal{T}[X, R_f]$. To prove that S is represented in G_2 exactly once we again show that S can be uniquely decomposed into three subpaths in G_2. Let P be the sibling of some piece $Q \in \mathcal{T}[X, R_f]$ s.t. S visits P, and let $p \in \partial P \cap S$. If P is not the deepest such piece, then S also visits Q but the edge (p, v) in G_2 counts only paths in $G \setminus Q$, hence S is not represented as a $u - p - v$ path in G_2. If P is the deepest such piece but p is not the first vertex in ∂P that S visits, then the u-to-p subpath of S is not represented as an edge (u, p) in G_2 since only paths in $G \setminus (P \cup Q) \cup \{p\}$ are.

3. S avoids all boundary vertices of all the pieces on $\mathcal{T}[X, R_f]$. S is not represented in G_1, G_2 because every path that is represented there touches some piece in $\mathcal{T}[X, R_f]$. It is counted exactly once in G_3 by a similar argument to case 2 above: S is counted once in G_3 by the first separator vertex that S visits in the rootmost piece that it visits. Finally, in the case where $R_u = R_v = R_f$, we perform Dijkstra (with its extension for counting) on $R_u \setminus \partial R_u$. $\qquad \square$

Efficient queries for planar graphs. We can easily achieve $\tilde{O}(n^{2/3})$ query-time, since this is the size of the graphs that we construct and can thus perform Dijkstra for SSSP computations. This query time applies to any graph family with $\sqrt{n}$-size separators, such as minor-free graphs. On planar graphs, in order to perform queries more efficiently we have to assume random access to the labels of vertices u, v and x; retrieving them would require $\tilde{O}(n^{2/3})$ time. We present an $\tilde{O}(\sqrt{n})$-time query algorithm for planar graphs at the expense of increasing the labels' size by polylogarithmic factors.

Let us now formally state the main result of [30].

Theorem 5 ([30])**.** *Given a weighted directed planar graph G of size n and a source $s \in V(G)$, we can construct in $O(n \log^4 n)$ time an $O(n \log^4 n)$-size data structure, that upon query (v, x), for $v, x \in V(G)$, returns the s-to-v distance in $G \setminus \{x\}$ in time $O(\log n)$.*

*C*ases 2 & 3. G_2 and G_3 are of size $O(\sqrt{n})$ and they can be constructed in $O(\sqrt{n})$ time from the labels of u, v and f. We can compute SSSPs in these graphs in $O(\sqrt{n} \log n)$ time using Dijkstra's algorithm. We handle the subcase of Case 3 in which $R_u = R_v$ and the sought shortest path does not cross ∂R_u as follows. The label of u additionally stores the single-source single-failure distance oracle of Theorem 5 for graph $R_u \setminus (\partial R_u \setminus \{u\})$ and source u. It occupies $\tilde{O}(r) = \tilde{O}(n^{2/3})$ additional space. Upon query, we simply query this oracle with (v, x).

*C*ase 1. This is the only involved case, as G_1 can be of size $\Theta(r) = \Theta(n^{2/3})$ and we aim at performing SSSP computations in time $\tilde{O}(\sqrt{n})$. Let us note that the distances of u to $\partial R_f \setminus \{f\}$ in the case that $R_u = R_f$ can be computed in time $\tilde{O}(\sqrt{r}) = \tilde{O}(n^{1/3})$ if we have stored the oracle of Theorem 5 for graph R_u and source u in the label of u. The case $R_v = R_f$ can be treated analogously.

In order to perform efficient SSSP computations we resort to FR-Dijkstra (Theorem 2). We first make a minor modification to item (iii) of the label so that the Monge property required for FR-Dijkstra is satisfied: instead of storing ∂R_u-to-∂R_u distances in $G \setminus \{u\}$, we instead store ∂R_u-to-∂R_u distances in $R_u \setminus \{u\}$ and ∂R_u-to-∂R_u distances in $G \setminus ((R_u \setminus \partial R_u) \cup \{u\})$. This ensures that the set of vertices over which the *DDG*s are built lie on a constant number of faces of the reference graph. The size of the label is unaffected by this modification. We can then use Theorem 2 in a straightforward way to compute the sought shortest path in time $\tilde{O}(\sqrt{r}) = \tilde{O}(n^{1/3})$.

Efficient preprocessing for planar graphs. The labels can be naively constructed in $O(n^2)$ time. This is true for any graph family with $\sqrt{n}$-size separators. For planar graphs, we show that the construction time can be improved to $\tilde{O}(n^{5/3})$.

The complete recursive decomposition of G, required for item (i), can be computed in $O(n)$ time [40]. For the rest of the items, we use MSSP data structure for an appropriate subgraph of G, or of the reverse graph of G, i.e. G with all its edges reversed.

The multiple-source shortest paths (MSSP) data structure [34] represents all shortest path trees rooted at the vertices of a single face g in a planar graph. It can be constructed in $O(n \log n)$ time, requires $O(n \log n)$ space, and can report any distance between a vertex of f and any other vertex in the graph in $O(\log n)$ time. Using a simple modification of the underlying graph, presented in [31], we can ensure that MSSP returns the length of the shortest path that is internally disjoint from a prespecified subset of the vertices of g.

To compute the information required for item (ii) of the labels, we build an MSSP data structure for the reverse graph of $G \setminus (R \setminus \partial R)$ for each piece R in the r-division and each of the $O(1)$ holes g on which the vertices of ∂R lie. We then query the sought distances. The time required to construct the MSSP data structures is $\tilde{O}(n^2/r) = \tilde{O}(n^{4/3})$ and the time required for computing the distances is $\tilde{O}(n^2/\sqrt{r}) = \tilde{O}(n^{5/3})$. The precomputations for items (iii), (iv) and the first part of item (v) can be done analogously –for item (iii) we store the distances described in the description of the efficient

query implementation.

For the second part of item (v), we can not make use of MSSP, as the shortest path from u to $p \in Sep(P)$ is allowed to cross $Sep(P)$. We can instead build an $\tilde{O}(|P|)$-size exact distance oracle for $P \setminus \partial P$ in $\tilde{O}(|P|^{3/2})$ time that answers distance queries in $\tilde{O}(|P|^{\epsilon})$ time, for any constant $\epsilon > 0$ ([45]); we pick $\epsilon = 1/6$. We then query this oracle for the all distances we need to compute in $P \setminus \partial P$. Over all pieces, the preprocessing time is $\tilde{O}(n^{3/2})$ and the sought distances are retrieved in $\tilde{O}(n^{3/2} \cdot n^{1/6}) = \tilde{O}(n^{5/3})$

To wrap up, the global preprocessing time is $\tilde{O}(n^{5/3})$ and is upper bounded by the total size of the labels up to polylogarithmic factors. The following theorem concludes this section.

Theorem 6. *Given a directed planar graph G of size n, with real edge-lengths, in overall $\tilde{O}(n^{5/3})$ time we can assign an $\tilde{O}(n^{2/3})$-size label to each vertex of G such that upon query (u, v, x), where $u, v, x \in V(G)$, the length of the shortest u-to-v path in $G \setminus \{x\}$ can be computed in $\tilde{O}(\sqrt{n})$ time from the labels of u, v and x.*

4. Labeling for Counting Shortest Paths

In this section we design labels such that given the labels of any $k + 2$ vertices $s, t, v_1, v_2, \ldots, v_k$, we should return the number of s-to-t paths that avoid vertices $v_1, \ldots, v_k$ and whose length is equal to $d(s, t)$ (the original s-to-t distance in G). Note that this is the same as returning the number of shortest s-to-t paths in $G \setminus \{v_1, v_2, \ldots, v_k\}$ only if the length of the shortest s-to-t path does not change when $\{v_1, v_2, \ldots, v_k\}$ fail. We show that the labeling of [37] (with labels of size $O(\sqrt{n})$) actually works in this more general setting and show how to perform a query in $\tilde{O}(\sqrt{n} \cdot k)$ time. We assume in this section that edge weights are positive. Formally, we show the following result:

Theorem 7. *Given a directed planar graph G of size n, with real positive edge-lengths and a positive integer k, we can assign a label to each vertex of G that consists of $O(\sqrt{n})$ machine words, each long enough to store the number of shortest paths between any two vertices in G, such that upon query $(s, t, v_1, v_2, \ldots, v_k)$, where $s, t, , v_1, v_2, \ldots, v_k \in V(G)$, the number of s-to-t paths in $G \setminus \{v_1, v_2, \ldots, v_k\}$ that have length equal to the distance from s to t in G can be computed in $\tilde{O}(\sqrt{n} \cdot k)$ time from the labels of $s, t, v_1, v_2, \ldots, v_k$.*

The label. We first compute a complete recursive decomposition of G. The label of each vertex u in G then consists of the following information:

15

(i) For each ancestor piece P of u, for every $v \in Sep(P)$, the number $p_1(u,v)$ and length $d_1(u,v)$ of all u-to-v shortest paths in $P \setminus Sep(P) \cup \{v\}$.

(ii) For each ancestor piece P of u, for every $v \in Sep(P)$, the number $p_2(v,u)$ and length $d_2(v,u)$ of all v-to-u shortest paths in $P \setminus \partial P$.

In what follows, in the case that v is in many separators of ancestor pieces of u, when referring to $d_1(u,v)$, $p_1(u,v)$, $d_2(v,u)$ and $p_2(v,u)$ we mean the values computed for the rootmost such piece.

The query - without faults. When there are no faulty vertices, every s-to-t shortest path Q_{st} in G is uniquely determined by a piece P in the recursive decomposition and a vertex $v \in Sep(P)$. The piece P is the rootmost ancestor piece of s in the recursive decomposition s.t. Q_{st} visits $Sep(P)$ and therefore does not visit ∂P. Such a piece P must be an ancestor of both s and t. The vertex $v \in Sep(P)$ is the first vertex of $Sep(P)$ visited by Q_{st}. Q_{st} can thus be decomposed into a prefix Q_1 in $P \setminus Sep(P) \cup \{v\}$ from s to v, and a suffix Q_2 in $P \setminus \partial P$ from v to t. For every possible v we have the number of such Q_1 in (i) of s and the number of such Q_2 in (ii) of t. We therefore add the term $p_1(s,v) \cdot p_2(v,t)$ to the answer. However, we only wish to add this term if $d(s,v) + d(v,t) = d(s,t)$ (otherwise, we are counting non-shortest paths). We have $d(s,v) + d(v,t)$ from the labels of s and t. We compute $d(s,t)$ as follows. Let $A[u]$ be the union of separator vertices of all ancestors of u. Then

$$d(s,t) = \min_{v \in A[s] \cap A[t]} (d_1(s,v) + d_2(v,t)), \tag{1}$$

and the overall query is computed as

$$paths(s,t) \quad = \sum_{\substack{v \in A[s] \cap A[t] \text{ s.t} \\ d_1(s,v)+d_2(v,t)=d(s,t)}} p_1(s,v) \cdot p_2(v,t) \tag{2}$$

It takes $\tilde{O}(\sqrt{n})$ time to perform such query because there are $O(\sqrt{n})$ vertices in $A[s] \cap A[t]$ and for each of them we perform $\tilde{O}(1)$ calculations. We also compute $d(s,t)$ beforehand in $\tilde{O}(\sqrt{n})$ time.

The query - with faults. We begin with an $\tilde{O}(\sqrt{n} \cdot k^2)$ time query and then improve this to $\tilde{O}(\sqrt{n} \cdot k)$. We order the faulty vertices in the increasing order of their distances from s in G, and index them $v_1, \ldots, v_k$ accordingly. For convenience we refer to s as v_0 and to t as v_{k+1}. Denote by $R[j]$ the number of s-to-v_j shortest paths in G that avoid $v_1, \ldots, v_{j-1}$. Denoting by $paths(v_i, v_j)$ the number

16

of v_i-to-v_j shortest paths in G we obtain the recurrence:

$$R[j] = paths(s, v_j) \; - \sum_{\substack{i < j \text{ s.t.} \\ d(s,v_i)+d(v_i,v_j)=d(s,v_j)}} R[i] \cdot paths(v_i, v_j) \tag{3}$$

To see why this recurrence holds, it suffices to show that every shortest path Q_{sv_j} in G from s to v_j that visits at least one of $v_1, \ldots, v_{j-1}$ is counted in the second term exactly once. It is clear that every such path is counted at least once, because it can be decomposed into a prefix composed of a shortest path from s to the first v_i that Q_{sv_j} visits (i.e. is counted by $R[i]$) and a suffix composed of a v_i-to-v_j path (i.e. counted by $paths(v_i, v_j)$). To see why every path Q_{sv_j} is counted at most once, notice that every such path visits the faulty vertices monotonically with respect to their ordering. In other words, if Q_{sv_j} visits some v_i and then some v_j then $i < j$. This holds because if v_i is on a shortest path from s to v_j then $d(s, v_i) < d(s, v_j)$, and by our ordering of the faulty vertices $i < j$. Since $R[i]$ only counts paths that are internally disjoint from failed vertices, the only time Q_{sv_j} is counted is when we count paths of the form $s \rightsquigarrow v_i \rightsquigarrow v_j$, where v_i is the first faulty vertex Q_{sv_j} visits.

Given $R[1], \ldots, R[j-1]$ we can compute $R[j]$ in $\tilde{O}(\sqrt{n} \cdot j)$ using the recurrence. For each faulty vertex v_i with $i < j$ we perform a $paths(v_i, v_j)$ query as described above which takes $\tilde{O}(\sqrt{n})$ time, so the overall complexity is $\tilde{O}(\sqrt{n} \cdot k^2)$.

Improved query time. We now show how to improve the query time from $\tilde{O}(\sqrt{n} \cdot k^2)$ to $\tilde{O}(\sqrt{n} \cdot k)$. In order to achieve this, we cannot afford to compute $paths(v_i, v_j)$ for every pair i, j. Instead, we will express $R[j]$ as a summation over $O(\sqrt{n})$ terms that we can compute in $\tilde{O}(1)$ time. By combining equations (2) and (3), and since $paths(s, v_j)$ can be computed in $\tilde{O}(\sqrt{n})$ time, we get that computing $R[j]$ boils down to computing the following double summation:

$$\sum_{\substack{i < j \text{ s.t.} \\ d(s,v_i)+d(v_i,v_j)=d(s,v_j)}} R[i] \sum_{\substack{v \in A[v_i] \cap A[v_j] \text{ s.t} \\ d_1(v_i,v)+d_2(v,v_j)=d(v_i,v_j)}} p_1(v_i, v) \cdot p_2(v, v_j) \tag{4}$$

The above sum counts all s-to-v_j shortest paths Q_{sv_j} that can be decomposed into three parts:

Q_1 - a shortest s-to-v_i path in G (for some v_i) that avoids $v_1, \ldots, v_{i-1}$.

Q_2 - a shortest v_i-to-v path in $P \setminus Sep(P) \cup \{v\}$ for some $v \in Sep(P)$, where P is defined as the rootmost ancestor of v_i s.t. Q touches $Sep(p)$ (v is the first vertex of $Sep(P)$ in Q_2).

Q_3 - a shortest v-to-v_j path in $P \setminus \partial P$.

17

We use the same decomposition into Q_1, Q_2, Q_3 but sum the terms differently. Denoting $D(s, v) = \min_i(d(s, v_i) + d_1(v_i, v))$ we compute:

$$\sum_{\substack{v \in A[v_j] \text{ s.t.} \\ D(s,v)+d_2(v,v_j)=d(s,v_j)}} p_2(v, v_j) \sum_{\substack{i<j \text{ s.t. } v \in A[v_i] \text{ and} \\ d(s,v_i)+d_1(v_i,v)=D(s,v)}} R[i] \cdot p_1(v_i, v) \tag{5}$$

Let us explain equation (5). Denote the inner summation term (in blue) as $F_j(v)$. $F_j(v)$ counts the number of combinations for $Q_1 Q_2$ by iterating over every faulty vertex v_i where $i < j$ and $v \in A[v_i]$. For a fixed v_i, the number of such combinations is $R[i] \cdot p_1(v_i, v)$. Among all $Q_1 Q_2$ combinations, we only want to sum combinations $Q_1 Q_2$ that have length $d(s, v)$. Ideally, this could be imposed by adding the condition $d(s, v_i) + d_1(v_i, v) = d(s, v)$ to the inner sum. However, we cannot compute $d(s, v)$ because we do not have the label of v. Instead, we add the condition $d(s, v_i) + d_1(v_i, v) = D(s, v)$ where $D(s, v) = \min_i(d(s, v_i) + d_1(v_i, v))$ (observe that $D(s, v) \geq d(s, v)$). This condition is easy to check using $d_1(v_i, v)$ stored in the label of v_i and the value $d(s, v_i)$ which can be computed beforehand using equation (1). The counting remains correct because in the outer sum we check that $D(s, v) + d_2(v, v_j) = d(s, v_j)$ which only holds if $D(s, v) = d(s, v)$ (because when $D(s, v) > d(s, v)$ then by the triangle inequality we have that $D(s, v) + d_2(v, v_j) > d(s, v) + d_2(v, v_j) \geq d(s, v_j)$). Note that even if $D(s, v) = d(s, v)$ it may be that $D(s, v) + d_2(v, v_j) > d(s, v_j)$. This happens in the case that there are no s-to-v_j shortest paths that visit v. In other words, we check that a path $Q_1 Q_2 Q_3$ is shortest by verifying that $d(s, v_i) + d_1(v_i, v) + d_2(v, v_j) = d(s, v_j)$. This is true iff $D(s, v) = d(s, v)$ and $d_2(v, v_j) = d(v, v_j)$ which means that $Q_1 Q_2 Q_3$ is indeed a shortest path.

Observe that in the inner sum we consider only $i < j$. This is because for $i \geq j$ none of the paths from s to v_j that visit v_i is shortest due to the ordering of the faulty vertices.

As for the outer sum, it counts the number of Q_3 paths for every $v \in A[v_j]$. Overall, we iterate over every $v \in A[v_j]$ and multiply $F_j(v)$ (the number of $Q_1 Q_2$ paths) by $p_2(v, v_j)$ (the number of Q_3 paths) and obtain the answer.

Overall, in the j-th iteration we compute $R[j]$ using the $F_j(v)$ values according to equation (5). Notice that $F_{j+1}(v)$ is either equal to $F_j(v)$ or to $F_j(v) + R[j] \cdot p_1(v_j, v)$. We can therefore compute $F_{j+1}(v)$ for every $v \in A[v_j]$ using the just computed $R[j]$ and $F_j(v)$. This takes total $\tilde{O}(\sqrt{n})$ time and $\tilde{O}(\sqrt{n} \cdot k)$ time over all the $k + 2$ iterations.

In order to check the distance restrictions in the summations we precompute $d(s, v_j)$ for every $0 \leq j \leq k + 1$ and $D(s, v)$ for every $v \in \bigcup_{0 \leq i \leq k+1} A[v_i]$. The former ($d(s, v_j)$) is computed using (1), and the latter ($D(s, v)$) is computed by iterating over every i and $v \in A[v_i]$ and maintaining the

18

minimum value for each $D(s, v)$. Thus, the precomputation of $D(s, v)$ and $d(s, v_j)$ takes $\tilde{O}(\sqrt{n} \cdot k)$ time.

5. A Lower Bound on Labels for Counting Shortest Paths

In this section we show a lower bound of $\Omega(\sqrt{nL})$ bits on the label-size for counting shortest paths (without faults) in undirected planar graphs, when the number of distinct shortest paths between any two vertices consists of at most L bits. The proof is a modification of the grid construction of Gavoille et al. [14] for standard distance labeling:

The grid construction of Gavoille et al. [14]. Consider a $(\sqrt{n}+1) \times (\sqrt{n}+1)$ grid consisting of vertices $v_{i,j}$, for all $i, j = 0, 1, \ldots, \sqrt{n}$ such that $i+j > 0$. For every $i = 1, 2, \ldots, \sqrt{n}$ and $j = 0, 1, \ldots, \sqrt{n} - 1$ we add an edge $(v_{i,j}, v_{i,j+1})$. For all $i, j = 1, 2, \ldots, \sqrt{n}$ we add an edge $(v_{i,j}, v_{i-1,j})$. The vertices $s_i = v_{i,0}$ for $i = 1, 2, \ldots, \sqrt{n}$ are called sources, and the vertices $t_j = v_{0,j}$ for $j = 1, 2, \ldots, \sqrt{n}$ are called targets. Then, for all $i, j = 1, 2, \ldots, \sqrt{n}$ we subdivide the edge $(v_{i,j-1}, v_{i,j})$ by inserting a new vertex $v'_{i,j}$, and similarly for all $i, j = 1, 2, \ldots, \sqrt{n}$ we subdivide the edge $(v_{i,j}, v_{i-1,j})$ by inserting a new vertex $v''_{i,j}$. We set the weight of every edge $(v_{i,j-1}, v'_{i,j})$ to be $n - i$, and the weight of every edge $(v''_{i,j}, v_{i-1,j})$ to be $n - j$. The weight of each other edge is 1.

It is easy to verify that the shortest path from the i-th source s_i to the j-th target t_j consists of $2j$ horizontal edges followed by $2i$ vertical edges. Moreover, one can encode any $\sqrt{n} \times \sqrt{n}$ boolean matrix M by inserting a (shortcut) edge (of weight 1) from $s = v'_{i,j}$ to $t = v''_{i,j}$ if and only if $M[i, j] = 1$. This way, the shortest s_i-to-t_j path is shorter by one (i.e. uses the shortcut edge) if and only if $M[i, j] = 1$. Since there are 2^n boolean matrices, the total label size of all sources and targets must be at least $\log(2^n) = n$ and hence some label must be of size $\Omega(\sqrt{n})$.

Theorem 8. *Any labeling scheme for counting shortest paths in planar graphs such that the number of distinct shortest paths between any two vertices consists of at most L bits requires labels consisting of $\Omega(\sqrt{nL})$ bits.*

Proof. Let us consider a $(\sqrt{m}+1) \times (\sqrt{m}+1)$ grid graph as in the above grid construction of Gavoille et al. [14]. In every intersection, instead of a single s-to-t shortcut, we introduce an $O(L)$-size gadget – essentially the one described in the proof of Fact 1.

More specifically, suppose that we are given $L - 1$ bits $b_0, \ldots, b_{L-2}$. Each edge of the gadget will have weight equal to $1/L$. The gadget consists of a path $s = u_0 - u_1 - \cdots - u_{L-1}$ and another path

$v_1 - v_2 - \cdots - v_L = t$ in which every edge is duplicated (i.e. there are two parallel edges between each pair v_i, v_{i+1}). Finally, for every $i = 0, \ldots, L-2$ such that $b_i = 1$, we add an edge $u_i - v_{i+1}$. The number of shortest s-to-t paths in the gadget is exactly $\sum_{i=0}^{L-2} b_i \cdot 2^{L-1-i}$. Note that this number is congruent to 0 modulo 2. The size of the graph is $n = \Theta(mL)$.

Now, the number of shortest paths from the i-th source vertex s_i to the j-th target vertex t_j is 1 if all b_i's are equal to 0 for the gadget at intersection (i, j); otherwise it is equal to $\sum_{i=0}^{L-2} b_i \cdot 2^{L-1-i}$. Hence, each pair (i, j) allows us to recover $\Theta(L)$ distinct bits. Thus, the total label size of all sources and targets must be at least $\log(2^{mL}) = mL$ and hence some label must be of size $\Omega(\sqrt{mL}) = \Omega(\sqrt{nL})$. $\qquad\square$

We leave the problem of closing the gap between this $\Omega(\sqrt{nL})$ lower bound and the $O(\sqrt{nL})$ upper bound open for further investigation.

6. A Lower Bound on Dynamic Oracles for Counting Shortest Paths

In this section we consider *dynamic oracles* for counting shortest paths (without faults) in undirected planar graphs. That is, data structures that can support queries for counting shortest paths as well as updates to the edge weights. We show a lower bound conditioned on the hardness of Online Boolean Matrix-Vector Multiplication (OMv):

Conjecture 9 (OMv Conjecture, [46]). *For every $\epsilon > 0$, there is no $O(N^{3-\epsilon})$-time algorithm that given an $N \times N$ boolean matrix M and a stream of boolean vectors $v_1, \ldots, v_N$ computes the products Mv_i online (i.e. computes Mv_i before seeing v_{i+1}).*

Based on the above conjecture, we prove that for any dynamic shortest paths counting oracle in undirected planar graphs, either the query time or the update time must be $\Omega(\sqrt{n})$ (up to subpolynomial factors).

Theorem 10. *A dynamic shortest paths counting oracle in undirected n-vertex planar graphs with amortized query time $q(n)$ and update time $u(n)$ cannot have $q(n) + u(n) = O(n^{1/2-\epsilon})$ for any $\epsilon > 0$ unless the OMv conjecture is false. This holds even if we only allow edge-weight increments and decrements by 1.*

Proof. Our proof follows closely the proof of Abboud-Dahlgaard [47] for dynamic oracles reporting (i.e. not counting) shortest paths. There are a few subtle differences, but the main difference

20

is that [47] was based on min-plus vector-matrix multiplication while ours is based on standard vector-matrix multiplication.

Encoding the matrix as a grid. We consider a $\sqrt{n} \times \sqrt{n}$ boolean matrix M (i.e. $N = \sqrt{n}$ in Conjecture 9) and encode it using a $(\sqrt{n}+1) \times (\sqrt{n}+1)$ grid G_M. For convenience, we index the rows and columns of M as $1, \ldots, \sqrt{n}$ and the rows and columns of G_M as $0, \ldots, \sqrt{n}$. The grid G_M contains:

1. All horizontal edges of the form $((i,j),(i,j+1))$ except for $((0,j),(0,j+1))$ (i.e. except for the first grid row). All these edges have the same weight $\sqrt{n}$.

2. All vertical edges of the form $((i,j),(i+1,j))$ except for $((i,\sqrt{n}),(i+1,\sqrt{n}))$ (i.e. except for the last grid column). The weight of edge $((i,j),(i+1,j))$ is $j+1$.

3. If $M_{i,j} = 1$ we add an edge $e_{i,j} = ((i-1,j-1),(i,j))$ with weight $\sqrt{n}+j$.

Denote the vertices of the first row $(0,j)$ as s_j and vertices of the last column $(i,\sqrt{n})$ as t_i. Consider the shortest s_j-to-t_i path. It is easy to see that if $M_{i,j+1} = 0$ then this path is (1) unique, (2) composed of a vertical prefix and and a horizontal suffix, and (3) is of length $\sqrt{n}(\sqrt{n}-j)+i(j+1)$. If however $M_{i,j+1} = 1$ then there are exactly two such shortest paths (one using $e_{i,j}$ and the other using $((i-1,j-1),(i,j-1))$ followed by $((i,j-1),(i,j)))$ both of length $\sqrt{n}(\sqrt{n}-j)+i(j+1)$.

The zero matrix grid. We would like to make the length of the above shortest paths independent of i and j. We define another $(\sqrt{n}+1) \times (\sqrt{n}+1)$ grid G_0 that has no diagonal edges and contains:

1. All horizontal edges of the form $((i,j),(i,j+1))$ except for $((0,j),(0,j+1))$ (i.e. except for the first grid row). All these edges have the same weight $\sqrt{n}$.

2. All vertical edges of the form $((i,j),(i+1,j))$ except for $((i,0),(i+1,0))$ (i.e. except for the first grid column). The weight of edge $((i,j),(i+1,j))$ is $\sqrt{n}-j+1$.

Denote the vertices of the first column $(i,0)$ of G_0 as s'_i and vertices of the first row $(0,j)$ of G_0 as t'_j. The graph G on which we build the oracle is obtained by connecting the two grids G_0 and G_{M^t} (the grid representation of the transpose of M). This is done by adding edges $b_i = (t_i, s'_i)$ of weight $w(b_i) = (\sqrt{n}+1)(\sqrt{n}-i)$ for every $1 \leq i \leq \sqrt{n}$.

The reduction. In order to solve the OMv problem, for each query vector $v[1, \ldots, \sqrt{n}]$, if v is the all-zero vector we simply output an all-zero vector. Otherwise, we (1) reset the weight $w(b_i)$ of every b_i to be $(\sqrt{n}+1)(\sqrt{n}-i)$, (2) for every i, if $v[i] = 0$ we increase the weight of b_i by 1, and (3) for every index $0 \leq j < \sqrt{n}$ we query the oracle for the number of shortest paths from s_j to t'_{j+1}. Finally, we decrease the oracle's answer by the number of 1's in v and assign this value as the j-th entry in the result Mv.

To see why the above procedure correctly calculates Mv, first note that the j-th entry in Mv is exactly the number of indices $1 \leq i \leq \sqrt{n}$ s.t. $e_{i,j}$ is present in G_{M^t} and $v[i] = 1$. The length of the shortest path from s_j to t'_{j+1} through an edge b_k is $d(s_j, t_k) + w(b_k) + d(s'_k, t'_{j+1}) = 2n + 2\sqrt{n}$. This value is independent of both j and k, so for each $1 \leq k \leq \sqrt{n}$ we have a unique shortest path through b_k if $e_{k,j+1}$ is absent in G or exactly two shortest paths if $e_{k,j+1}$ is present in G. In step (2), when we increase by 1 the edges b_i corresponding to entries in v where $v[i] = 0$, paths going through these b_i's are longer than paths going through other b_i's and are therefore not shortest (we made sure that v is not all-zero). Hence, every b_i that corresponds to $v[i] = 1$ contributes $1 + M^t_{i,j+1}$ to the number of shortest paths from s_j to t'_{j+1}, and by subtracting the number of 1's in v we obtain the correct answer.

Overall, for each vector we perform $O(\sqrt{n})$ updates and queries, so overall we perform $O(n)$ updates and queries. If each update/query takes $O(n^{0.5-\epsilon})$ time then we get overall $O(n^{1.5-\epsilon}) = O(N^{3-\epsilon/2})$ contradicting Conjecture 9. □

7. Conclusions and Open Problems

Our main contribution in this paper is a single-fault distance labeling scheme for planar graphs with label size $\tilde{O}(n^{2/3})$, query time $\tilde{O}(\sqrt{n})$, and construction time $\tilde{O}(n^{5/3})$.

The main open question that is left unanswered by our work is the existence of non-trivial multiple-fault distance labels (tolerating more than a single fault). Labels for approximate distances [26] also rely on separators, and do handle multiple failures. In the failure-free case, the labels of [26] consist of distances to a small (logarithmic) sample of vertices on some separators, called connections. To handle failures, the label of each vertex u also stores the failure-free labels of the connections of u. This only increases the label-size by a polylogarithmic factor. In case of exact distances, the size of the failure-free labels is $\Omega(\sqrt{n})$, so this approach seems unsuitable.

Another natural open question is whether the gap between our $O(n^{2/3})$-size fault-tolerant labels and the $O(n^{1/2})$-size labels without failures is actually required and tight. We observe that the existing lower bound technique of Gavoille et al. cannot be extended to show a lower bound above $\Omega(\sqrt{n})$ for fault-tolerant labels. The reason is that their technique uses a global argument showing that if we wish to encode the distances between a subset S of $k \leq \sqrt{n}$ vertices then all their labels together require size $\Omega(k^2)$. However, even in the presence of (any number of) failures, encoding distances can be done with total size $\tilde{O}(k^2)$ (simply store for every $u, v \in S$ the length of the shortest u-to-v path that is internally disjoint from S).

References

[1] S. Kannan, M. Naor, S. Rudich, Implicit representation of graphs, SIAM Journal on Discrete Mathematics 5 (4) (1992) 596–603. `doi:10.1137/0405049`.

[2] S. Alstrup, S. Dahlgaard, M. B. T. Knudsen, Optimal induced universal graphs and adjacency labeling for trees, J. ACM 64 (4) (2017) 27:1–27:22. `doi:10.1145/3088513`.

[3] C. Petersen, N. Rotbart, J. G. Simonsen, C. Wulff-Nilsen, Near-optimal adjacency labeling scheme for power-law graphs, in: 43rd ICALP, 2016, pp. 133:1–133:15. `doi:10.4230/LIPIcs.ICALP.2016.133`.

[4] S. Alstrup, H. Kaplan, M. Thorup, U. Zwick, Adjacency labeling schemes and induced-universal graphs, SIAM J. Discret. Math. 33 (1) (2019) 116–137. `doi:10.1137/16M1105967`.

[5] N. Alon, R. Nenadov, Optimal induced universal graphs for bounded-degree graphs, in: 28th SODA, 2017, pp. 1149–1157. `doi:10.1137/1.9781611974782.74`.

[6] N. Bonichon, C. Gavoille, A. Labourel, Short labels by traversal and jumping, Electronic Notes in Discrete Mathematics 28 (2007) 153–160. `doi:10.1016/j.endm.2007.01.022`.

[7] M. Katz, N. A. Katz, A. Korman, D. Peleg, Labeling schemes for flow and connectivity, SIAM J. Comput. 34 (1) (2004) 23–40. `doi:10.1137/S0097539703433912`.

[8] T. Hsu, H. Lu, An optimal labeling for node connectivity, in: 20th ISAAC, 2009. `doi:10.1007/978-3-642-10631-6\_32`.

[9] A. Korman, Labeling schemes for vertex connectivity, ACM Trans. Algorithms 6 (2) (2010) 39:1–39:10. `doi:10.1145/1721837.1721855`.

[10] D. Peleg, Informative labeling schemes for graphs, Theor. Comput. Sci. 340 (3) (2005) 577–593. `doi:10.1016/j.tcs.2005.03.015`.

[11] N. G. Rotbart, New ideas on labeling schemes, Ph.D. thesis, University of Copenhagen (2016).

[12] R. L. Graham, H. O. Pollak, On embedding graphs in squashed cubes, in: Y. Alavi, D. R. Lick, A. T. White (Eds.), Graph Theory and Applications, Springer Berlin Heidelberg, Berlin, Heidelberg, 1972, pp. 99–110.

[13] D. Peleg, Proximity-preserving labeling schemes, J. Graph Theory 33 (3) (2000) 167–176.

[14] C. Gavoille, D. Peleg, S. Pérennes, R. Raz, Distance labeling in graphs, Journal of Algorithms 53 (1) (2004) 85–112. `doi:10.1016/j.jalgor.2004.05.002`.

[15] P. Gawrychowski, P. Uznański, Better distance labeling for unweighted planar graphs, in: 17th WADS, 2021, pp. 428–441. `doi:10.1007/978-3-030-83508-8\_31`.

[16] C. Gavoille, M. Katz, N. A. Katz, C. Paul, D. Peleg, Approximate distance labeling schemes, in: 9th ESA, 2001, pp. 476–487. `doi:10.1007/3-540-44676-1\_40`.

[17] A. Gupta, A. Kumar, R. Rastogi, Traveling with a pez dispenser (or, routing issues in MPLS), in: 42nd FOCS, 2001, pp. 148–157. `doi:10.1109/SFCS.2001.959889`.

[18] M. Thorup, Compact oracles for reachability and approximate distances in planar digraphs, Journal of the ACM 51 (6) (2004) 993–1024. `doi:10.1145/1039488.1039493`.

[19] I. Abraham, C. Gavoille, Object location using path separators, in: 25th PODC, ACM, 2006, pp. 188–197. `doi:10.1145/1146381.1146411`.

[20] J. Feigenbaum, D. R. Karger, V. S. Mirrokni, R. Sami, Subjective-cost policy routing, in: 1st WINE, 2005, pp. 174–183. `doi:10.1007/11600930\_18`.

[21] J. Feigenbaum, D. R. Karger, V. S. Mirrokni, R. Sami, Subjective-cost policy routing, Theor. Comput. Sci. 378 (2) (2007) 175–189. `doi:10.1016/j.tcs.2007.02.020`.

[22] A. D. Twigg, Compact forbidden-set routing, Tech. Rep. UCAM-CL-TR-678, University of Cambridge, Computer Laboratory (Dec. 2006).
URL https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-678.pdf

[23] B. Courcelle, C. Gavoille, M. M. Kanté, Compact labelings for efficient first-order model-checking, Journal of Combinatorial Optimization 21 (1) (2009) 19–46. doi:10.1007/s10878-009-9260-7.

[24] B. Courcelle, A. Twigg, Constrained-path labellings on graphs of bounded clique-width, Theory of Computing Systems 47 (2) (2010) 531–567. doi:10.1007/s00224-009-9211-9.

[25] I. Abraham, S. Chechik, C. Gavoille, D. Peleg, Forbidden-set distance labels for graphs of bounded doubling dimension, ACM Trans. Algorithms 12 (2) (2016) 22:1–22:17. doi:10.1145/2818694.

[26] I. Abraham, S. Chechik, C. Gavoille, Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels, in: 44th STOC, 2012, pp. 1199–1218. doi:10.1145/2213977.2214084.

[27] Y. Emek, D. Peleg, L. Roditty, A near-linear-time algorithm for computing replacement paths in planar directed graphs, ACM Trans. Algorithms 6 (4) (2010) 64:1–64:13. doi:10.1145/1824777.1824784.

[28] P. N. Klein, S. Mozes, O. Weimann, Shortest paths in directed planar graphs with negative lengths: A linear-space $O(n \log^2 n)$-time algorithm, ACM Trans. Algorithms 6 (2) (2010) 30:1–30:18. doi:10.1145/1721837.1721846.

[29] C. Wulff-Nilsen, Solving the replacement paths problem for planar directed graphs in $O(n \log n)$ time, in: 21st SODA, 2010, pp. 756–765. doi:10.1137/1.9781611973075.62.

[30] S. Baswana, U. Lath, A. S. Mehta, Single source distance oracle for planar digraphs avoiding a failed node or link, in: 23rd SODA, 2012, pp. 223–232. doi:10.1137/1.9781611973099.20.

[31] P. Charalampopoulos, S. Mozes, B. Tebeka, Exact distance oracles for planar graphs with failing vertices, ACM Trans. Algorithms (2022). doi:10.1145/3511541.

[32] G. F. Italiano, A. Karczmarz, N. Parotsidis, Planar reachability under single vertex or edge failures, in: 32nd SODA, 2021, pp. 2739–2758. doi:10.1137/1.9781611976465.163.

[33] J. Fakcharoenphol, S. Rao, Planar graphs, negative weight edges, shortest paths, and near linear time, J. Comput. Syst. Sci. 72 (5) (2006) 868–889. `doi:10.1016/j.jcss.2005.05.007`.

[34] P. N. Klein, Multiple-source shortest paths in planar graphs, in: 16th SODA, 2005, pp. 146–155. URL `http://dl.acm.org/citation.cfm?id=1070432.1070454`

[35] G. F. Italiano, Y. Nussbaum, P. Sankowski, C. Wulff-Nilsen, Improved algorithms for min cut and max flow in undirected planar graphs, in: 43rd STOC, 2011, pp. 313–322. `doi:10.1145/1993636.1993679`.

[36] H. Kaplan, S. Mozes, Y. Nussbaum, M. Sharir, Submatrix maximum queries in monge matrices and partial monge matrices, and their applications, ACM Trans. Algorithms 13 (2) (2017) 26:1–26:42. `doi:10.1145/3039873`.

[37] I. Bezáková, A. Searns, On counting oracles for path problems, in: 29th ISAAC, 2018, pp. 56:1–56:12. `doi:10.4230/LIPIcs.ISAAC.2018.56`.

[38] S. Mozes, C. Wulff-Nilsen, Shortest paths in planar graphs with real lengths in $O(n \log^2 n / \log \log n)$ time, in: 18th ESA, 2010, pp. 206–217. `doi:10.1007/978-3-642-15781-3\_18`.

[39] G. L. Miller, Finding small simple cycle separators for 2-connected planar graphs, in: 16th STOC, 1984, pp. 376–382. `doi:10.1145/800057.808703`.

[40] P. N. Klein, S. Mozes, C. Sommer, Structured recursive separator decompositions for planar graphs in linear time, in: 45th STOC, 2013, pp. 505–514. `doi:10.1145/2488608.2488672`.

[41] G. N. Frederickson, Fast algorithms for shortest paths in planar graphs, with applications, SIAM J. Comput. 16 (6) (1987) 1004–1022. `doi:10.1137/0216064`.

[42] S. Cabello, E. W. Chambers, J. Erickson, Multiple-source shortest paths in embedded graphs, SIAM J. Comput. 42 (4) (2013) 1542–1571. `doi:10.1137/120864271`.

[43] G. Monge, Mémoire sur la théorie des déblais et des remblais, De l'Imprimerie Royale, 1781.

[44] P. Gawrychowski, S. Mozes, O. Weimann, C. Wulff-Nilsen, Better tradeoffs for exact distance oracles in planar graphs, in: 29th SODA, 2018, pp. 515–529. `doi:10.1137/1.9781611975031.34`.

[45] P. Charalampopoulos, P. Gawrychowski, S. Mozes, O. Weimann, Almost optimal distance oracles for planar graphs, in: 51st STOC, 2019, pp. 138–151. `doi:10.1145/3313276.3316316`.

[46] M. Henzinger, S. Krinninger, D. Nanongkai, T. Saranurak, Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture, in: 47th STOC, 2015, pp. 21–30. `doi:10.1145/2746539.2746609`.

[47] A. Abboud, S. Dahlgaard, Popular conjectures as a barrier for dynamic planar graph algorithms, in: 57th FOCS, 2016, pp. 477–486. `doi:10.1109/FOCS.2016.58`.