Exact Distance Oracles for Planar Graphs with Failing Vertices*

Panagiotis Charalampopoulos† Shay Mozes‡ Benjamin Tebeka§

Abstract

We consider exact distance oracles for directed weighted planar graphs in the presence of failing vertices. Given a source vertex u, a target vertex v and a set X of k failed vertices, such an oracle returns the length of a shortest u-to-v path that avoids all vertices in X. We propose oracles that can handle any number k of failures. More specifically, for a directed weighted planar graph with n vertices, any constant k, and for any $q \in [1, \sqrt{n}]$, we propose an oracle of size $\tilde{\mathcal{O}}(\frac{n^{k+3/2}}{q^{2k+1}})$ that answers queries in $\tilde{\mathcal{O}}(q)$ time.[1] In particular, we show an $\tilde{\mathcal{O}}(n)$-size, $\tilde{\mathcal{O}}(\sqrt{n})$-query-time oracle for any constant k. This matches, up to polylogarithmic factors, the fastest failure-free distance oracles with nearly linear space. For single vertex failures ($k = 1$), our $\tilde{\mathcal{O}}(\frac{n^{5/2}}{q^3})$-size, $\tilde{\mathcal{O}}(q)$-query-time oracle improves over the previously best known tradeoff of Baswana et al. [SODA 2012] by polynomial factors for $q = \Omega(n^t)$, $t \in (1/4, 1/2]$. For multiple failures, no planarity exploiting results were previously known.

1 Introduction

Computing shortest paths is one of the most well-studied algorithmic problems. In the data structure version of the problem, the aim is to compactly store information about a graph such that the distance (or the shortest path) between any queried pair of vertices can be retrieved efficiently. Data structures supporting distance queries are called *distance oracles*. The two main measures of efficiency of a distance oracle are the space it occupies and the time it requires to answer a distance query. Another quantity of interest is the time it takes to construct the oracle.

In recent decades researchers have investigated the shortest path problem in graphs subject to failures, or more broadly, to changes. One such variant is the *replacement paths problem.* In this problem we are given a graph G and vertices u and v. The goal is to report the u-to-v distance in G for each possible failure of a single edge along the shortest u-to-v path. Another variant is that of constructing a distance oracle that answers u-to-v distance queries subject to edge or vertex failures (u, v and the set of failures are given at query time). Perhaps the most general of these variants is the *fully-dynamic* distance oracle; a data structure that supports distance queries as well as updates to the graph such as changes to edge lengths, edge insertions or deletions and vertex insertions or deletions.

One obvious but important application of handling failures is in geographical routing. Further motivation for studying this problem originates from Vickrey pricing in networks [31, 21]; see [10] for a concise discussion on the relation between the problems. A long-studied generalization of the shortest path problem is the the k-shortest path, in which not one but but several shortest paths must be produced between a pair of vertices. This problem reduces to running k executions of a replacement paths algorithm, and has many applications itself [14].

In this paper we focus on these problems, and in particular on handling vertex failures in planar graphs. Observe that edge failures easily reduce to vertex failures. Indeed, by replacing each edge (a, c) of G with a new dummy vertex b and appropriately weighted edges (a, b) and (b, c); the failure of edge (a, c) in G corresponds to the failure of vertex b in the new graph. Note that this transformation does not depend on planarity. In sparse graphs, such as planar graphs, this transformation only increases the number of vertices by a constant factor. Also note that there is no such obvious reduction in the other direction that preserves planarity. In general graphs, one can replace each vertex v by two vertices v_{in} and v_{out}, assign to v_{in} (resp. v_{out}) all the edges incoming to v (resp. outgoing from v) and add a 0-length directed edge e from v_{in} to v_{out}. The failure of vertex v in the original graph corresponds to the failure of edge e in the new graph. However, this transformation does not preserve planarity.

*This work was partially supported by Israeli Science Foundation (ISF) grants 794/13 and 592/17.

†Department of Informatics, King's College London, London, UK, panagiotis.charalampopoulos@kcl.ac.uk, Efi Arazi School of Computer Science, The Interdisciplinary Center Herzliya, Herzliya, Israel

‡Efi Arazi School of Computer Science, The Interdisciplinary Center Herzliya, Herzliya, Israel, smozes@idc.ac.il

§Efi Arazi School of Computer Science, The Interdisciplinary Center Herzliya, Herzliya, Israel, benitbk@gmail.com

[1] The $\tilde{\mathcal{O}}(\cdot)$ notation hides polylogarithmic factors.

1.1 Related Work

General Graphs. Demetrescu et al. presented an $\mathcal{O}(n^2 \log n)$-size oracle answering single failure distance queries in constant time [10]. Bernstein and Karger, improved the construction time in [5]. Interestingly, Duan and Pettie, building upon this work, showed an $\mathcal{O}(n^2 \log^3 n)$-size oracle that can report distances subject to two failures, in time $\mathcal{O}(\log n)$ [12]. Based on this oracle, they then easily obtain an $\tilde{\mathcal{O}}(n^k)$-space oracle answering distance queries in $\tilde{\mathcal{O}}(1)$ time for any $k \geq 2$. Oracles that require less space for more than 2 failures have been proposed, such as the one presented in [32], but at the expense of $\Omega(n)$ query time. Such oracles are unsatisfactory for planar graphs, where single source shortest paths can be computed in linear or nearly linear time.

Planar Graphs. Exact (failure-free) distance oracles for planar graphs have been studied extensively over the past three decades [11, 3, 8, 16, 29, 6, 9, 20]. The known space to query-time tradeoffs have been significantly improved very recently [20, 9]. The currently best known tradeoff is an oracle of size $\tilde{\mathcal{O}}(n^{3/2}/q)$, that answers queries in time $\tilde{\mathcal{O}}(q)$ for any $q \in [1, n^{1/2}]$ [20]. Note that all known oracles with nearly linear (i.e. $\tilde{\mathcal{O}}(n)$) space require $\Omega(\sqrt{n})$ query time.

As for handling failures, the replacement paths problem (i.e. when both the source and destination are fixed in advance) can be solved in nearly linear time [13, 26, 33]. For the single source, single failure version of the problem (i.e. when the source vertex is fixed at construction time, and the query specifies just the target and a single failed vertex), Baswana et al. [4] presented an oracle with size and construction time $\mathcal{O}(n \log^4 n)$ that answers queries in $\mathcal{O}(\log^3 n)$ time. They then showed an oracle of size $\tilde{\mathcal{O}}(n^2/q)$ for the general single failure problem (i.e. when the source, destination, and failed vertex are all specified at query time), that answers queries in time $\tilde{\mathcal{O}}(q)$ for any $q \in [1, n^{1/2}]$. They conclude the paper by asking whether it is possible to design a compact distance oracle for a planar digraph which can handle multiple vertex failures. We answer this question in the affirmative.

Fakcharoenphol and Rao, in their seminal paper [16], presented distance oracles that require $\mathcal{O}(n^{2/3} \log^{7/3} n)$ and $\mathcal{O}(n^{4/5} \log^{13/5} n)$ amortized time per update and query for non-negative and arbitrary edge-weight updates respectively.[2] The space required by these oracles is $\mathcal{O}(n \log n)$. Klein presented a similar data structure in [24] for the case where edge-weight updates are non-negative, requiring time $\mathcal{O}(n^{2/3} \log^{5/3} n)$. Klein's result was extended in [22], where, assuming non-negativity of edge-weight updates, the authors showed how to handle edge deletions and insertions (not violating the planarity of the embedding), and in [23], where the authors showed how to handle negative edge-weight updates, all within the same time complexity. In fact, these results can all be combined, and along with a recent slight improvement on the running time of FR-Dijkstra [19], they yield a dynamic distance oracle that can handle any of the aforementioned edge updates and queries within time $\mathcal{O}(n^{2/3} \frac{\log^{5/3} n}{\log^{4/3} \log n})$. We further extend these results by showing that vertex deletions and insertions can also be handled within the same time complexity. The main challenge lies in handling vertices of high degree.

For the case where one is willing to settle for approximate distances, Abraham et al. [2] gave a $(1+\epsilon)$ labeling scheme for undirected planar graphs with polylogarithmic size labels, such that a $(1+\epsilon)$-approximation of the distance between vertices u and v in the presence of $|F|$ vertex or edge failures can be recovered from the labels of u, v and the labels of the failed vertices in $\tilde{\mathcal{O}}(|F|^2)$ time. They then use this labeling scheme to devise a fully dynamic $(1+\epsilon)$-distance oracle with size $\tilde{\mathcal{O}}(n)$ and $\tilde{\mathcal{O}}(\sqrt{n})$ query and update time.[3]

On the lower bounds side, it is known that an exact dynamic oracle requiring amortized time $\mathcal{O}(n^{1/2-\delta})$, for any constant $\delta > 0$, for both edge-weight updates and distance queries, would refute the APSP conjecture, i.e. that there is no truly subcubic combinatorial algorithm for solving the all-pairs shortest path problems in weighted (general) graphs [1].

1.2 Our Results and Techniques

In this work we focus on distance queries subject to vertex failures in planar graphs. Our results can be summarized as follows.

1. We show how to preprocess a directed weighted planar graph G in $\tilde{\mathcal{O}}(n)$ time into an oracle of size $\tilde{\mathcal{O}}(n)$ that, given a source vertex u, a target vertex v, and a set X of k failing vertices, reports the length of a shortest u-to-v path in $G \setminus X$ in $\tilde{\mathcal{O}}(\sqrt{kn})$ time. See Lemma 3.2.
2. For k allowed failures, and for any $r \in [1, n]$, we show how to construct an $\tilde{\mathcal{O}}(\frac{n^{k+1}}{r^{k+1}}\sqrt{nr})$-size oracle that answers queries in time $\tilde{\mathcal{O}}(k\sqrt{r})$. See Theorem 4.1. For $k = 1$, this improves over the previously best known tradeoff of Baswana et al. [4] by polynomial factors for $r = \Omega(n^t)$, $t \in (1/2, 1]$.

[2] Though this is not mentioned in [16], the query time can be made worst case rather than amortized by standard techniques.

[3] A fully dynamic distance oracle supports arbitrary edge and vertex insertions and deletions, and length updates.

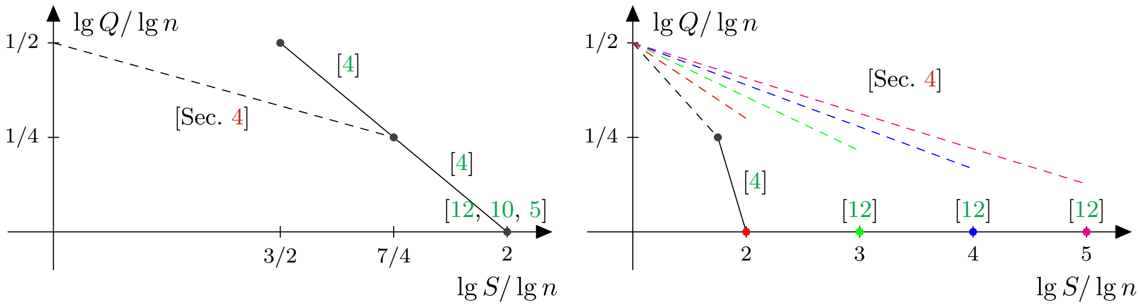


Figure 1: Left: Tradeoff of the Space (S) vs. the Query time (Q) for exact distance oracles for a single failed vertex (i.e. $k = 1$) on a doubly logarithmic scale, ignoring constant and logarithmic factors. The previous tradeoff is indicated by a solid line, while the new tradeoff is indicated by a dashed line. Right: the same tradeoff for $k = 1, \ldots, 5$, shown with different colours. The points on the x-axis correspond to the result of [12], while the new tradeoffs are indicated by dashed lines.

To the best of our knowledge, this is the first tradeoff for $k > 1$. See Fig. 1.

3. We extend the exact dynamic distance oracles mentioned in the previous section to also handle vertex insertions and deletions without changing their space and time bounds.

Our nearly-linear space oracle that reports distances in the presence of k failures in $\tilde{\mathcal{O}}(\sqrt{kn})$ time is obtained by adapting a technique of Fakcharoenphol and Rao [16]. In a nutshell, a planar graph can be recursively decomposed using small cycle separators, such that the boundary of each piece in the decomposition (i.e. the vertices of a piece that also belong to other pieces in the decomposition) is a cycle with relatively few vertices. Instead of working with the given planar graph, one computes distances over its *dense distance graph* (DDG); a non-planar graph on the boundary vertices of the pieces which captures the distances between boundary vertices within each of the underlying pieces. Fakcharoenphol and Rao developed an efficient implementation of Dijkstra's algorithm on the DDG. This algorithm, nicknamed *FR-Dijkstra*, runs in time roughly proportional to the number of vertices of the DDG (i.e. boundary vertices), rather than in time proportional to the number of vertices in the planar graph. Roughly speaking, Fakcharoenphol and Rao show that to obtain distances from u to v with k edge failures, it (roughly) suffices to consider just the boundary vertices of the pieces in the recursive decomposition that contain failed edges. Since pieces at the same level of the recursive decomposition are edge-disjoint, the total number of boundary vertices in all the required pieces is only $\mathcal{O}(\sqrt{kn})$. This $\tilde{\mathcal{O}}(n)$-size, $\tilde{\mathcal{O}}(\sqrt{kn})$-query-time oracle, supporting distance queries subject to a batch of k edge cost updates, leads to their dynamic distance oracle.

The difficulty in handling vertex failures is that a high degree vertex x may be a boundary vertex of many (possibly $\Omega(n)$) pieces in the recursive decomposition. Then, if x fails, one would have to consider too many pieces and too many boundary vertices. Standard techniques such as degree reduction by vertex splitting are inappropriate because when a vertex fails all its copies fail. To overcome this difficulty we define a variant of the dense distance graph which, instead of capturing shortest path distances between boundary vertices within a piece, only captures distances of paths that are internally disjoint from the boundary. We show that such distances can be computed efficiently, and that it then suffices to include in the FR-Dijkstra computation (roughly) only pieces that contain x, but not as a boundary vertex. This leads to our nearly-linear-space oracle reporting distances in the presence of k failures in $\tilde{\mathcal{O}}(\sqrt{kn})$ time (item 1 above). See Section 3. Plugging the same technique into the existing dynamic distance oracles extends them to support vertex deletions (item 3 above). See Section 6.

Our main result, the space vs. query-time tradeoff (item 2 above), is obtained by a non-trivial combination of this technique with ideas from the recent static distance oracle presented in [20]. Namely, by a combination of FR-Dijkstra on our variant of the DDG with r-divisions, external *DDG*s, and efficient point location in Voronoi diagrams. See Sections 4 and 5.

2 Preliminaries

In this section we review the main techniques required for describing our result. Throughout the paper we consider a weighted directed planar graph $G =$

$(V(G), E(G))$, embedded in the plane. (We use the terms weight and length for edges and paths interchangeably throughout the paper.) We use $|G|$ to denote the number of vertices in G. Since planar graphs are sparse, $|E(G)| = \mathcal{O}(|G|)$ as well. For an edge (u, v), we say that u is its tail and v is its head. $d_G(u, v)$ denotes the distance from u to v in G. We denote by $d_G(u, v, X)$ the distance from u to v in $G \setminus X$, where $X \in V(G)$ or $X \subset V(G)$. If the reference graph is clear from the context we may omit the subscript. We assume that the input graph has no negative length cycles. If it does, we can detect this in $\mathcal{O}(n\frac{\log^2 n}{\log\log n})$ time by computing single source shortest paths from any vertex [30]. In the same time complexity, we can transform the graph in a standard way so that all edge weights are non-negative and shortest paths are preserved. We further assume that shortest paths are unique as required for a result from [18] that we use; this can be ensured in $\mathcal{O}(n)$ time by a deterministic perturbation of the edge weights [15]. Each original distance can be recovered from the corresponding distance in the transformed graph in constant time.

Separators and recursive decompositions in planar graphs. Miller [27] showed how to compute a Jordan curve that intersects the graph at $\mathcal{O}(\sqrt{n})$ nodes and separates it 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 [25] show how to obtain a complete recursive decomposition tree $\mathcal{T}$ of G in $\mathcal{O}(n)$ time. $\mathcal{T}$ is a binary tree whose nodes correspond to subgraphs of G (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 $P \in \mathcal{T}$. An r-division [17] of a planar graph, for $r \in [1, n]$, is a decomposition of the graph into $\mathcal{O}(n/r)$ pieces, each of size $\mathcal{O}(r)$, such that each piece has $\mathcal{O}(\sqrt{r})$ boundary vertices, i.e. vertices incident to edges in other pieces. Another usually desired property of an r-division is that the boundary vertices lie on a constant number of faces of the piece (holes). For every r larger than some constant, an r-division with this property (i.e. few holes per piece) is represented in the decomposition tree $\mathcal{T}$ of [25]. Throughout the paper, to avoid confusion, we use "nodes" when referring to $\mathcal{T}$ and "vertices" when referring to G. We denote the boundary vertices of a piece P by ∂P. We refer to non-boundary vertices as internal.

LEMMA 2.1. ([20]) *Each node in $\mathcal{T}$ corresponds to a piece such that (i) each piece has $\mathcal{O}(1)$ holes, (ii) the number of vertices in a piece at depth ℓ in $\mathcal{T}$ is $\mathcal{O}(n/c_1^\ell)$, for some constant $c_1 > 1$, (iii) the number of boundary vertices in a piece at depth ℓ in $\mathcal{T}$ is $\mathcal{O}(\sqrt{n}/c_2^\ell)$, for some constant $c_2 > 1$.*

We use the following well-known bounds (see e.g., [20]).

PROPOSITION 2.1. *$\sum_{P\in\mathcal{T}} |P| = \mathcal{O}(n \log n)$, $\sum_{P\in\mathcal{T}} |\partial P| = \mathcal{O}(n)$ and $\sum_{P\in\mathcal{T}} |\partial P|^2 = \mathcal{O}(n \log n)$.*

We show the following bound that will be used in future proofs.

PROPOSITION 2.2. *$\sum_{P\in\mathcal{T}} |P||\partial P|^2 = \mathcal{O}(n^2)$.*

Proof. Let $P_1^\ell, P_2^\ell, \ldots, P_j^\ell$ be the pieces at the ℓ-th level of the decomposition. $\sum_i |P_i^\ell| = \mathcal{O}(n)$ since the pieces are edge-disjoint. We know by Lemma 2.1 that $|\partial P_j^\ell| \le \sqrt{n}/c_2^\ell$ for all j and hence $|\partial P_j^\ell|^2 \le n/c_2^{2\ell}$ for all j. It follows that $\sum_i |P_i^\ell||\partial P_i^\ell|^2 = \mathcal{O}(n^2/c_2^{2\ell})$ and the claimed bound follows by summing over all levels of $\mathcal{T}$. □

Dense distance graphs and FR-Dijkstra. The *dense distance graph* of a piece P, denoted DDG_P is a complete directed graph on the boundary vertices of P. Each edge (u, v) has weight $d_P(u, v)$, equal to the length of the shortest u-to-v path in P. DDG_P can be computed in time $\mathcal{O}(|\partial P|^2 + |P| \log |P|)$ using the multiple source shortest paths (MSSP) algorithm [24, 7]. Over all pieces of the recursive decomposition this takes time $\mathcal{O}(n \log^2 n)$ in total and requires space $\mathcal{O}(n \log n)$ by Proposition 2.1. We next give a —convenient for our purposes— interface for FR-Dijkstra [16], 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_P satisfy the Monge property. (A matrix M satisfies the Monge property if, for all $i < i'$ and $j < j'$, $M_{i,j} + M_{i',j'} \le M_{i',j} + M_{i,j'}$ [28].) The interface is specified in the following theorem, which was essentially proved in [16], with some additional components and details from [23, 30].

THEOREM 2.1. ([16, 23, 30]) *A set of DDGs with $\mathcal{O}(M)$ vertices in total (with multiplicities), each having at most m vertices, can be preprocessed in time and space $\mathcal{O}(M \log m)$ in total. After this preprocessing, Dijkstra's algorithm can be run on the union of any subset of these DDGs with $\mathcal{O}(N)$ vertices in total (with multiplicities) in time $\mathcal{O}(N \log^2 m)$, by relaxing edges in batches. Each such batch consists of edges that have the same tail.*

Voronoi diagrams with point location. Let P be a directed planar graph with real edge-lengths, and no negative-length cycles. Let S be a set of vertices that lie on a single face of P; we call the elements of S sites. Each site $u \in S$ has a weight $\omega(s) \geq 0$ associated with it. The additively weighted distance between a site $s \in S$ and a vertex $v \in V$, denoted by $d^{\omega}_P(s, v)$ is defined as $\omega(s)$ plus the length of the s-to-v shortest path in P.

Definition 2.1. *The additively weighted Voronoi diagram of (S, ω) ($VD(S, \omega)$) within P is a partition of $V(P)$ into pairwise disjoint sets, one set $\mathsf{Vor}(s)$ for each site $s \in S$. The set $\mathsf{Vor}(s)$ which is called the Voronoi cell of s, contains all vertices in $V(P)$ that are closer (w.r.t. $d^{\omega}_P(. \;, .)$) to s than to any other site in S (assuming that the distances are unique). There is a dual representation $VD^*(S, \omega)$ of a Voronoi diagram $VD(S, \omega)$ as a planar graph with $\mathcal{O}(|S|)$ vertices and edges.*

Theorem 2.2. ([20, 18]) *Given subsets $S'_1, \ldots, S'_m$ of S, and additive weights $\omega_i(u)$ for each $u \in S'_i$, we can construct a data structure of size $\mathcal{O}(|P| \log |P| + \sum_i |S'_i|)$ that supports the following (*point location*) queries. Given i, and a vertex v of P, report in $\mathcal{O}(\log^2 |P|)$ time the site s in the additively weighted Voronoi diagram $VD(S_i, \omega_i)$ such that v belongs to $\mathsf{Vor}(s)$ and the distance $d^{\omega_i}_P(s, v)$. The time and space required to construct this data structure are $\tilde{\mathcal{O}}(|P||S|^2 + \sum_i |S'_i|)$.*

Remark. Part of Theorem 2.2 is proved in [20], though not stated there explicitly as a theorem. It is a tradeoff to Theorem 1.1 of [20], requiring less space, and hence more applicable to our problem.

3 Near linear space data structure for any number of failures

In this section we show how to adapt the approach of [16] for dynamic distance oracles supporting cumulative edge changes to support distance queries with failed vertices. The main technical challenge is in dealing with failures of high-degree vertices, since such vertices may belong to many pieces at each level of the decomposition. For example, think of a failure of the central vertex in a wheel graph, which belongs to all the pieces in the recursive decomposition. Note that standard degree reduction techniques such as vertex splitting are not useful because when a vertex fails all its copies fail. This is in contrast with the situation when dealing only with edge-weight updates, since each edge can be in at most one piece per level. We circumvent this by defining and employing the *strictly internal dense distance graph*.

Definition 3.1. *The* strictly internal *dense distance graph of a piece P, denoted DDG°_P, is a complete directed graph on the boundary vertices of P. An edge (u, v) has weight $d^{\circ}_P(u, v)$ equal to the length of the shortest u-to-v path in P that is internally disjoint from ∂P.*

The sole difference to the standard Definition is that in our case paths are not allowed to go through ∂P. Observe that the shortest path in P between two vertices of ∂P is still represented in DDG°_P, just not necessarily by a single edge as in DDG_P. This establishes the following lemma.

Lemma 3.1. *For any piece P and any two boundary vertices $u, v \in \partial P$, the u-to-v distance in DDG°_P equals the u-to-v distance in DDG_P.*

We now discuss how to efficiently compute DDG°_P. We construct a planar graph $\hat{P}$, by creating a copy of P and incrementing the weight of each edge uv, such that $u \in \partial P$, by $C = 2\sum_{e \in E(G)} |w(e)|$. $DDG_{\hat{P}}$ can be computed in $\mathcal{O}(|\partial P|^2 + |P| \log |P|)$ time using MSSP [24, 7]. Observe that any u-to-v path in $\hat{P}$ that starts at $\partial \hat{P}$ and is internally disjoint from $\partial \hat{P}$ has exactly one edge uw with $u \in \partial P$, so its length is at least C and less than $2C$, while any u-to-v path that has an internal vertex in ∂P is of length at least $2C$. Therefore, the u-to-v distance in $\hat{P}$ is equal to C plus the length of the shortest u-to-v path in P that is internally disjoint from ∂P if the latter one is not ∞. We thus set $d^{\circ}_P(u, v) = d_{\hat{P}}(u, v) - C$. This completes the description of the computation of DDG°_P. Note that since C is defined in terms of G rather than P, edge weights greater than C in DDG°_P effectively represent infinite length in the sense that such edges will never be used by any shortest path (in P nor in G). Also note that it follows directly from the Definition of the Monge property that subtracting C from each entry of a Monge matrix preserves the Monge property. Therefore, we can use $\bigcup_P DDG^{\circ}_P$ in FR-Dijkstra (Theorem 2.1) instead of $\bigcup_P DDG_P$.

Preprocessing. We compute a complete recursive decomposition tree $\mathcal{T}$ of G in time $\mathcal{O}(n)$ as discussed in Section 2. We compute DDG°_P for each non-leaf piece $P \in \mathcal{T}$ and preprocess it as in FR-Dijkstra. By Proposition 2.1, Theorem 2.1 and the above discussion, the time and space complexities are $\mathcal{O}(n \log^2 n)$ and $\mathcal{O}(n \log n)$ respectively.

Query. Upon query (u, v, X), for each $i \in \{u, v\} \cup X$ we arbitrarily choose a leaf-piece P_i containing i, and run FR-Dijkstra on the union of the following DDG°s, which we denote by $\mathcal{D}$ (inspect Fig. 2 for an illustration):

1. For each $w \in \{u, v\}$, $DDG^\circ_{P_w}$ of $P_w \setminus X$ with w regarded as a boundary vertex. This can be computed on the fly in constant time since the size of the leaf piece P_w is constant.

2. For each $w \in \{u, v\}$, for each strict ancestor piece P of P_w in $\mathcal{T}$, DDG°_P if P does not contain an internal (i.e. non-boundary) vertex of X.

3. For each $x \in X$, $DDG^\circ_{P_x}$ of $P_x \setminus X$. This can be computed on the fly in constant time since the size of the leaf piece P_x is constant.

4. For each $x \in X$, for each ancestor P of P_x (including P_x), DDG°_Q of the sibling Q of P if Q does not contain an internal vertex of X.

We can identify these DDG°s in $\mathcal{O}(k \log n)$ time by traversing the parent pointers from each P_i, for $i \in X$, and marking all the nodes that have an internal failed vertex. We make one small but crucial change to FR-Dijkstra. When running FR-Dijkstra, we do not relax edges whose tail is a failed vertex. This guarantees that, although failed vertices might appear in the graph on which FR-Dijkstra is invoked, the u-to-v shortest path computed by FR-Dijkstra does not contain any failed vertices. We therefore obtain the following lemma.

LEMMA 3.2. *There exists a data structure of size $\mathcal{O}(n \log n)$, which can be constructed in $\mathcal{O}(n \log^2 n)$ time, and answer the following queries in $\mathcal{O}(\sqrt{kn} \log^2 n)$ time. Given vertices u and v, and a set X of k failing vertices, report the length of a shortest u-to-v path in that avoids the vertices of X.*

Proof. We have already discussed the space occupied by the oracle and the time required to build it. It remains to analyze the query algorithm.

Correctness. First, it is easy to see that no edge (y, z) of any of the DDG°s in $\mathcal{D}$ represents a path containing a vertex $x \in X$, unless $\{y, z\} \cap X \neq \emptyset$. The latter case does not affect the correctness of the algorithm, since in FR-Dijkstra we do not relax edges whose tail is a failed vertex. Hence, the algorithm never computes a distance corresponding to a path going through a failed vertex.

It remains to show that the shortest path in $G \setminus X$ is represented in $\mathcal{D}$. For this, it suffices to prove that for each ancestor A of P_u (and similarly of P_v), either DDG°_A for $A \setminus X$ belongs to $\mathcal{D}$, or $\mathcal{D}$ contains enough information to reconstruct DDG°_A for $A \setminus X$ (i.e. subject to the failures) during FR-Dijkstra. In the latter case we say that DDG°_A is *represented* in $\mathcal{D}$. Note that, for any piece P, DDG°_P is represented in $\mathcal{D}$ if the DDG°s of its two children in $\mathcal{T}$ are represented in $\mathcal{D}$. If A contains no internal failed vertex then DDG°_A is in $\mathcal{D}$ by point 1 or 2 above. We next consider the case that A does contain some failed vertex $x \in X$ as an internal vertex. Thus A is an ancestor of P_x. To show that A is represented in $\mathcal{D}$, we prove that for any failed vertex $y \in X$, the DDG° of any ancestor of P_y in $\mathcal{T}$ is represented in $\mathcal{D}$.

We proceed by the minimal counterexample method. For any $x \in X$, $DDG^\circ_{P_x}$ of the leaf piece $P_x \in X$ is in $\mathcal{D}$ since it is computed on the fly in point 3. Let F be the deepest node in $\mathcal{T}$ that contains a failed vertex and whose DDG° subject to the failures is not represented in $\mathcal{D}$. Since F contains some failed vertex y, at least one of F's children in $\mathcal{T}$ contains a failed vertex. If both children of F in $\mathcal{T}$ contain failed vertices, then by the choice of the deepest such F, the DDG°s of both children of F are represented in $\mathcal{D}$, and therefore, so does DDG°_F, a contradiction. If, on the other hand, one of child of F, say K, contains a failed vertex and the other, say J, does not, we have DDG°_K represented in $\mathcal{D}$ by the choice of F as deepest, and DDG°_J in $\mathcal{D}$ by point 4 (J is a sibling of K with no internal failed vertices). Thus, DDG°_F is represented in $\mathcal{D}$, which is again a contradiction.

Time complexity. Let $r = n/k$ and consider an r-division of G in $\mathcal{T}$. The pieces of this r-division have $\mathcal{O}(\frac{n}{\sqrt{r}}) = \mathcal{O}(\sqrt{kn})$ boundary vertices in total and this is known to also be an upper bound on the total number of boundary vertices (with multiplicities) of ancestors of pieces in this r-division (cf. the discussion after Corollary 5.1 in [20]).

Recall that we have chosen a leaf-piece P_i for each vertex $i \in \{u, v\} \cup X$. It is easy to see that all the pieces whose DDG°s belong to $\mathcal{D}$ are either ancestors of some P_i or siblings of such a node. This implies that each $i \in \{u, v\} \cup X$ contributes the DDG°s of at most two pieces per level of the decomposition. Let the ancestor of P_i that is in the r-division be R_i. For each P_i, we only need to bound the total size of pieces it contributes that are descendants of R_i, since we have already bounded the total size of the rest. We do so by applying Lemma 2.1 for the subtree of $\mathcal{T}$ rooted at each R_i. (The extra $\mathcal{O}(\sqrt{r})$ boundary vertices we start with do not alter the analysis of this lemma as these many are anyway introduced by the first separation of R_i.) It yields $2\sum_\ell \frac{\sqrt{r}}{c_2^\ell}$, where $c_2 > 1$, which is $\mathcal{O}(\sqrt{r})$. Summing over all $k + 2$ pieces P_i we obtain the upper bound $\mathcal{O}(k\sqrt{r}) = \mathcal{O}(\sqrt{kn})$.

FR-Dijkstra runs in time proportional to the total number of vertices of the DDG°s in $\mathcal{D}$ up to a $\log^2 n$ multiplicative factor and hence the time complexity follows. ☐

Remark. By using existing techniques (cf. [23, Section 5.4]), we can report the actual shortest path ρ in time $\mathcal{O}(|\rho| \log\log \Delta_\rho)$, where Δ_ρ is the maximum degree of a vertex of ρ in G.[4]

4 Tradeoffs

In this section we describe a tradeoff between the size of the oracle and the query-time. We first define another useful modification of dense distance graphs.

DEFINITION 4.1. *The strictly external dense distance graph of G for pieces $P_1, \dots, P_i$ ($DDG^\circ_{ext}(P_1, \dots, P_i)$) is a complete directed graph on the boundary vertices of $P_1, \dots, P_i$. The edge (u, v) has weight equal to the length of the shortest u-to-v path in $G \setminus \big((\bigcup_{j=1}^{i} P_j) \setminus \{u, v\}\big)$.*

DDG°_{ext}s can be preprocessed using Theorem 2.1 together with DDG°s so that we can perform efficient Dijkstra computations in any union of DDG°_{ext}s and DDG°s.

The number of pieces in an r-division is at most cn/r for some constant c. For convenience, we define

$$g(n, r, k) = \binom{cn/r}{k} \le \frac{(cn)^k}{r^k k!} \le \frac{n^k}{r^k k},$$

where the last inequality holds for sufficiently large k. We use $g(n, r, k)$ throughout to encapsulate the dependency on k.

4.1 The case of a single failure

For ease of presentation we first describe an oracle that can handle just a single failure. We prove the following lemma, which is a restricted version of our main result, Theorem 4.1.

LEMMA 4.1. *For any $r \in [1, n]$, there exists a data structure of size $\mathcal{O}(\frac{n^{5/2}}{r^{3/2}} + n \log^2 n)$, which can be constructed in time $\tilde{\mathcal{O}}(\frac{n^{5/2}}{r^{3/2}} + n^2)$, and can answer the following queries in $\mathcal{O}(\sqrt{r} \log^2 n)$ time. Given vertices u, v, x, report the length of a shortest u-to-v path that avoids x.*

We first perform the precomputations of Section 3. We also obtain an r-division of G from $\mathcal{T}$ in $\mathcal{O}(n)$ time. Let us denote the pieces of this r-division by $R_1, \dots, R_q$.

[4]This remark also applies to the dynamic distance oracle presented in Section 6. However, it does not apply to the oracles presented in Section 4, where we use a different modification of DDGs for which we can not afford to store the MSSP data structures that would allow us to return the actual shortest paths efficiently.

Warm up. We first show how to get an $\mathcal{O}(\frac{n^3}{r^2})$-space oracle with $\tilde{\mathcal{O}}(\sqrt{r})$ query time for a single failure using the approach of Section 3. For each triplet R_i, R_j, R_k of pieces in the r-division we store $DDG^\circ_{ext}(R_i, R_j, R_k)$; these require space $\mathcal{O}(g(n, r, 3)(\sqrt{r})^2) = \mathcal{O}(\frac{n^3}{r^2})$ in total. Given u, v, x in R_u, R_v and R_x, respectively, we consider the required DDG°s that allow us to represent $DDG^\circ_{R_j}$ subject to the failures for each j as in Section 3 (i.e. the DDG°s in items 2 and 4 in Section 3 are only taken for ancestors of P_i that are descendants of R_j). We then run FR-Dijkstra on these along with $DDG^\circ_{ext}(R_u, R_v, R_x)$, not relaxing edges whose tail is x if encountered. This takes time $\mathcal{O}(\sqrt{r} \log^2 n)$.

Main Idea for reducing the space complexity. Instead of storing information for triplets of pieces, we will store more information, but just for pairs. Given u, v, x we show how to compute $d(u, v, x)$ relying on the information stored for the pair of pieces R_u and R_x. We first compute the distances from u to each $w \in \partial R_u \cup \partial R_x$ in $G \setminus \{x\}$ using FR-Dijkstra with $DDG^\circ_{ext}(R_u, R_x)$ as in the warm up above. We then identify an appropriate piece Q in $\mathcal{T}$ that contains v, and does not contain u nor x. Exploiting the fact that distances within Q remain unchanged when x fails, we employ Voronoi Diagrams with point location for the piece Q, adapting ideas from [20].

Additional Preprocessing. For each pair of pieces (R_i, R_j) of the r-division we compute and store the following:

1. $DDG^\circ_{ext}(R_i, R_j)$.
2. Let S be a separator in the recursive decomposition, separating a piece into two subpieces Q and R, such that $R_i \subseteq R$ and $R_j \not\subseteq Q$. For each $y \in \partial R_i \cup \partial R_j$, for each hole h of Q, we compute and store a Voronoi diagram with the point location data structure for Q, with sites the boundary vertices of Q that lie on h, and additive weights the distances from y to these sites in $G \setminus ((R_i \cup R_j) \setminus \{y\})$.

We now show that the space required is $\mathcal{O}(\frac{n^{5/2}}{r^{3/2}} + n \log^2 n)$. The space required for the preprocessed internal and external dense distance graphs is $\mathcal{O}(n \log n)$ and $\mathcal{O}(\frac{n^2}{r})$, respectively, by Theorem 2.1. We next analyze the space required for storing the Voronoi diagrams. We consider $\mathcal{O}(g(n, r, 2)) = \mathcal{O}(\frac{n^2}{r^2})$ pairs of pieces (R_i, R_j), and for each of the $\mathcal{O}(\sqrt{r})$ boundary vertices of each such pair we store, in the worst case, a Voronoi diagram for each of the $\mathcal{O}(1)$ holes of each sibling of the nodes in the root-to-R_i and root-to-R_j paths in $\mathcal{T}$. The total number of sites of all Voronoi

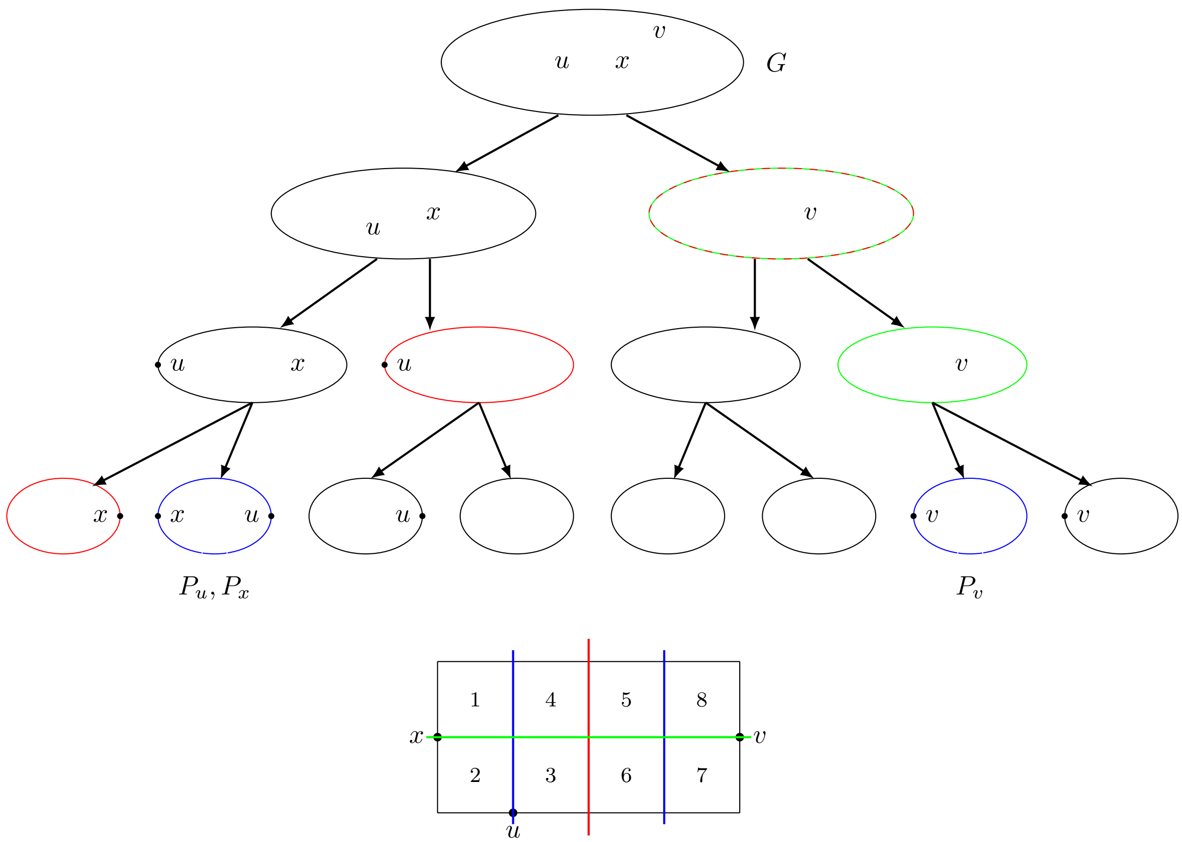


Figure 2: On top: The view of a recursive decomposition tree $\mathcal{T}$. With blue the pieces whose DDG°s are considered by points 1 and 3, with green the ones by point 2 and with red the ones by point 4. Note that a DDG° can be added for multiple reasons, e.g. the right child of the root must be included by both points 2 and 4. On the bottom: A grid graph. The vertices of each cell correspond to a leaf-piece in $\mathcal{T}$; the number in each cell is the rank of the leaf it corresponds to in a left-to-right ordering of the leaves. The separator of G is shown in red, the separators of each of children in blue, and the ones at the lowest level in green.

diagrams we store for a pair of pieces can be upper bounded by $\mathcal{O}(\sqrt{n})$ by noting that the number of sites at level ℓ of T_G has $\mathcal{O}(\sqrt{n}/c_2^{\ell})$ boundary vertices by Lemma 2.1. By Theorem 2.2, the space required to store a representation of a set of Voronoi diagrams with the functionality allowing for efficient point location queries for a piece P, with sites a subset of the boundary vertices of P, lying on a hole h is $\mathcal{O}\big(\sum_{P\in\mathcal{T}}(\mathcal{S}_{P,h} + |P|\log|P|)\big)$, where $\mathcal{S}_{P,h}$ is the total cardinality of these sets of sites. Summing over all holes of all pieces P, noting that $\sum_{P\in\mathcal{T}}\sum_h \mathcal{S}_{P,h} = \mathcal{O}(\frac{n^{5/2}}{r^{3/2}})$ by the above discussion, and using Proposition 2.1, the total space required for all Voronoi diagrams is $\mathcal{O}(\frac{n^{5/2}}{r^{3/2}} + n\log^2 n)$.

We analyze the construction time in Section 5. The internal dense distance graphs can be computed in time $\mathcal{O}(n\log^2 n)$. The external dense distance graphs and the additive weights can be computed in time $\mathcal{O}(\frac{n^2}{r}\log^2 n)$ and $\mathcal{O}(\frac{n^2}{r}\sqrt{nr}\log^3 n)$, respectively; see Lemmas 5.1 and 5.2. We show in Lemma 5.3 that we can compute all required Voronoi diagrams in time $\tilde{\mathcal{O}}(n^2 + \mathcal{S})$, where $\mathcal{S}$ is the size of their representation described in Section 2.

Query. If any two of $\{u, v, x\}$ are in the same piece of the r-division, then we can use FR-Dijkstra taking into account just two pieces of the r-division containing u, v, and x, similarly to the description in the warm up above. We therefore assume no two of $\{u, v, x\}$ are in the same piece of the r-division. We first retrieve a piece R_v of the r-division, containing v (to support that, each vertex stores a pointer to some piece of the r-division that contains it). In the following we will need to check whether a vertex is in some particular piece of $\mathcal{T}$. This can be done in $\mathcal{O}(\log n)$ time by storing, for each piece in $\mathcal{T}$, a binary tree with the vertices in the piece. We then proceed as follows (inspect Fig. 3 for an illustration).

1. Following parent pointers of R_v in $\mathcal{T}$, we find the highest ancestor Q of R_v containing neither u nor x. Thus, the sibling R of Q in $\mathcal{T}$ contains a vertex $i \in \{u, x\}$. We find a descendant R_i of R that is in the r-division and contains i. We then find any piece R_j of the r-division containing the element of $\{u, x\} \setminus \{i\}$. Note that, by choice of Q, R_j is not a descendant of Q. Finding these pieces requires time $\mathcal{O}(\log^2 n)$.

2. Let P_u be a leaf descendant of R_u in $\mathcal{T}$ that contains u. We run FR-Dijksta (not relaxing edges whose tail is x if encountered) on:

   (a) DDG°s of each of the ancestors of P_u that is a descendant of R_u in $\mathcal{T}$, including P_u and R_u;

   (b) $DDG^\circ_{ext}(R_u, R_x)$;

   (c) the DDG°s that allow us to represent $DDG^\circ_{R_x}$ subject to the failure of x as in Section 3.

   This takes time $\mathcal{O}(\sqrt{r}\log^2 n)$ and returns $d_G(u, y, x)$ for each $y \in \partial R_u \cup \partial R_x$.

3. For each $y \in (\partial R_u \cup \partial R_x) \setminus \{x\}$, for each hole h of Q, we perform an $\mathcal{O}(\log^2 n)$-time query to the Voronoi diagram stored for R_u, R_x, y, and h to get the distance from y to v in $G \setminus ((R_u \cup R_x) \setminus \{y\})$. The required distance is the minimum $d_G(u, y, x) + d(y, v, (R_u \cup R_x) \setminus \{y\})$ over all y. Each query takes $\mathcal{O}(\log^2 n)$ time and hence the total time required is $\mathcal{O}(\sqrt{r}\log^2 n)$.

We now argue the correctness of the query algorithm. Let ρ be a shortest u-to-v path that avoids x. Let z be the last vertex of ρ that belongs to $\partial R_u \cup \partial R_x$. Let h' be the hole of Q such that the last vertex of ρ that belongs to the boundary of Q belongs to hole h'. The distance $d_G(u, z, x)$ from u to z in $G \setminus \{x\}$ is computed by the FR-Dijkstra computation in step 2, while the distance from z to v in $G \setminus \{x\}$ is obtained from the query to the Voronoi diagram stored for R_u, R_x, z, and h'. It is easy to see that we do not obtain any distance that does not correspond to an actual path in $G \setminus \{x\}$ and hence the correctness of the query algorithm follows.

4.2 Handling multiple failures The warm-up approach of Section 4.1 can be trivially generalized to handle k failed vertices by considering $(k + 2)$-tuples of pieces of the r-division. (We consider the elements of tuples to be unordered throughout.) The space required is $\tilde{\mathcal{O}}(g(n, r, k + 2)(\sqrt{r})^2) = \tilde{\mathcal{O}}(\frac{n^{k+2}}{r^{k+1}})$ and queries can be answered in $\tilde{\mathcal{O}}(k\sqrt{r})$ time. We reduce the space to $\tilde{\mathcal{O}}(\frac{n^{k+1}}{r^{k+1}}\sqrt{nr})$ by generalizing the main algorithm of Section 4.1.

Preprocessing.

1. We perform the precomputations of Section 3.

2. For each $(k+1)$-tuple of pieces $(R_{i_1}, \ldots, R_{i_{k+1}})$ of the r-division we compute and store the following:

   (a) $DDG^\circ_{ext}(R_{i_1}, \ldots, R_{i_{k+1}})$.

   (b) Let S be a separator in the recursive decomposition, separating a piece into Q and R, such that for some j $R_{i_j} \subseteq R$ and none of the other pieces of the tuple is a subgraph of Q. For each $y \in \bigcup_{j=1}^{k+1} \partial R_{i_j}$, for each hole h of Q, we store a Voronoi diagram with the point location data structure for Q, with sites the boundary vertices of Q that lie on h, and additive weights the distances from y to these sites in $G \setminus ((\bigcup_{j=1}^{k+1} R_{i_j}) \setminus \{y\})$.

Query. We first retrieve a piece R_v of the r-division, containing v. We can again assume that no two elements of $\{u\} \cup X$ are in the same piece of the r-division, since otherwise we can answer the query in $\mathcal{O}(k\sqrt{r})$ time by running FR-Dijkstra on the DDG°_{ext} of a $(k+1)$-tuple and the DDG°s we add for each of the pieces in the tuple, following the algorithm of Section 3.

The algorithm is then essentially the same as that of Section 4.1.

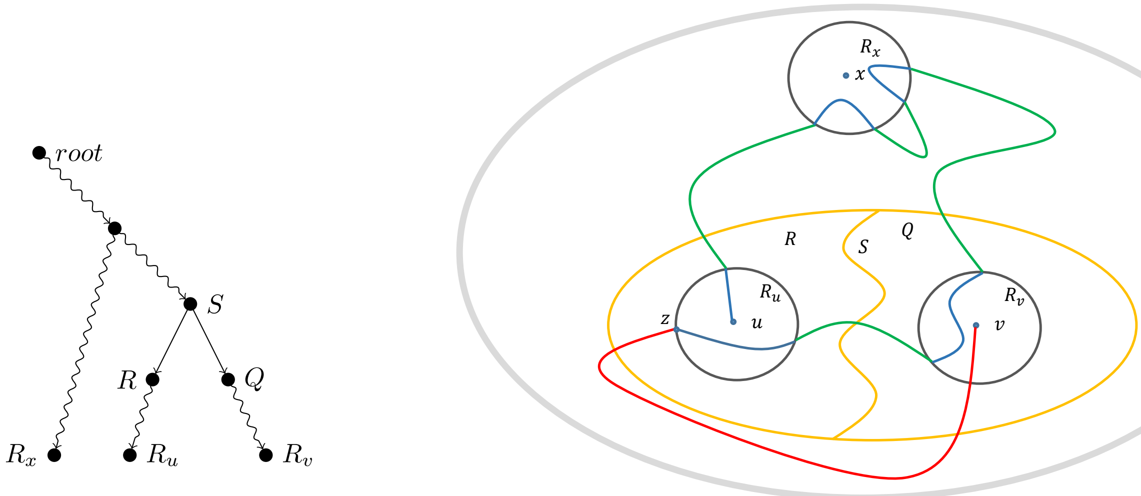
1. We find the highest ancestor Q of R_v in $\mathcal{T}$ that does not contain any of the elements of $\{u\} \cup X$ and retrieve a descendant of its sibling in the r-division that does contain some element $i \in \{u\} \cup X$. We then identify a piece R_j in the r-division for each $j \in \{u\} \cup X \setminus \{i\}$. This requires time $\mathcal{O}(k \log^2 n)$.

2. We run FR-Dijkstra on DDG°s of total size $\mathcal{O}(k\sqrt{r})$.

3. We perform $\mathcal{O}(k\sqrt{r})$ point location queries to Voronoi diagrams of Q, each requiring time $\mathcal{O}(\log^2 n)$.

We hence obtain the general tradeoff theorem.

THEOREM 4.1. *For any integer $r \in [1, n]$ and for any integer $k \le \frac{n}{r}$, there exists a data structure of size $\mathcal{O}(\frac{(cn)^{k+1}}{r^{k+1}}\frac{1}{k!}\sqrt{nkr} + n\log^2 n)$, which can be constructed in time $\tilde{\mathcal{O}}(\frac{(cn)^{k+1}}{r^{k+1}}\frac{1}{k!}\sqrt{nkr} + n^2)$, for some constant $c > 1$, and can answer the following queries in $\mathcal{O}(k\sqrt{r}\log^2 n)$ time. Given vertices u and v and a set X of at most k failing vertices, report the length of a shortest u-to-v path that avoids X.*



(a) root-to-R_i paths in $\mathcal{T}$ (b) The u-to-v path in $G \setminus \{x\}$

Figure 3: To the left: A view of the root-to-R_i paths in $\mathcal{T}$. Straight edges denote edges of the tree, while snake-shaped edges denote paths. To the right: A view of the shortest path in G. The paths in blue are represented by the DDG°s, the ones in green by DDG°_{ext} and the length of the one in red is returned by the point location query in the Voronoi diagram.

Remark. Our distance oracle can handle any number f of failures that lie in at most k pieces of the r-division in time $\tilde{\mathcal{O}}((k+\sqrt{fk})\sqrt{r})$ with an $\tilde{\mathcal{O}}(\frac{n^{k+1}}{r^{k+1}}\sqrt{nr})$-size oracle. This follows from the fact that the DDG°s we will add for a piece with f_i failures have total size $\tilde{\mathcal{O}}(\sqrt{f_i r})$ by the same analysis as in the proof of Lemma 3.2 and the fact that, given $f_1, \ldots, f_k$ such that $\sum_{i=1}^{k} f_i = f$, we have $\sum_{i=1}^{k} \sqrt{f_i} \leq \sqrt{fk}$ by the Cauchy-Schwarz inequality.

Proof. [of Theorem 4.1.] The correctness of the query algorithm follows by an argument identical to the one for the case of single failures (see Section 4.1); its time complexity is analyzed above. We next analyze the space required by our data structure and its construction time.

Space Complexity. The space occupied by the preprocessed DDG°s and DDG°_{ext}s is $\mathcal{O}(n \log n)$ and $\mathcal{O}(g(n,r,k+1)k^2 r) = \mathcal{O}(\frac{(cn)^{k+1}}{r^{k+1}}\frac{kr}{k!})$, respectively, by Theorem 2.1.

We bound the space required for the Voronoi diagrams by $\mathcal{O}(g(n,r,k+1)k\sqrt{nkr} + n\log^2 n)$ as follows. For each of the $\mathcal{O}(k\sqrt{r})$ boundary vertices of each of the $\mathcal{O}(g(n,r,k+1))$ $(k+1)$-tuples, we store a Voronoi diagram for each of the $\mathcal{O}(1)$ holes, of (at most) each of the siblings of the nodes in the root-to-R_i path in $\mathcal{T}$ for each R_i in the tuple. With an argument identical to the one used in the proof of Theorem 2.1, the total number of boundary vertices (with multiplicities) of all of these pieces is $\mathcal{O}(\sqrt{kn})$. Hence the total number of all Voronoi diagrams that we store is $\mathcal{O}(g(n,r,k+1)k\sqrt{nkr})$. By Theorem 2.2, the size required to store them, with the required functionality, is $\mathcal{O}(g(n,r,k+1)k\sqrt{nkr} + \sum_{P\in\mathcal{T}} |P|\log|P|) = \mathcal{O}(\frac{(cn)^{k+1}}{r^{k+1}}\frac{1}{k!}\sqrt{nkr} + n\log^2 n)$, where the last equality follows by Proposition 2.1.

The total space is thus $\mathcal{O}(\frac{(cn)^{k+1}}{r^{k+1}}\frac{1}{k!})(kr+\sqrt{nkr}) + n\log^2 n) = \mathcal{O}(\frac{(cn)^{k+1}}{r^{k+1}}\frac{1}{k!}\sqrt{nkr} + n\log^2 n)$ since $k \leq n/r$.

Preprocessing time. We compute the DDG°_{ext}s and the required additive weights of all $(k+1)$-tuples in time $\tilde{\mathcal{O}}(\frac{(cn)^{k+1}}{r^k}\frac{1}{(k-1)!})$ and $\tilde{\mathcal{O}}(\frac{(cn)^{k+1}}{r^{k+1}}\frac{1}{(k-1)!}\sqrt{nkr})$, respectively, using Lemmas 5.1 and 5.2. Finally, constructing the Voronoi diagrams requires time $\tilde{\mathcal{O}}(n^2 + \mathcal{S})$, where $\mathcal{S}$ is the total size of their representation, which is equal to the total number of sites in these diagrams (with multiplicities), as shown in Lemma 5.3; this dominates the time complexity. ☐

5 Efficient preprocessing

In this section we show how to efficiently compute the data structures described in Section 4. It is shown in [25, Theorem 3] that, given a geometrically increasing sequence of numbers $\nabla = (r_1, r_2, \dots, r_\nu)$, where r_1 is a sufficiently large constant, $r_{i+1}/r_i = b$, for all i, for some constant $b > 1$, and $r_\nu = n$, we can obtain r-divisions for all $r \in \nabla$ in time $\mathcal{O}(n)$ in total. These r-divisions satisfy the property that a piece in the r_i-division is a —not necessarily strict— descendant (in $\mathcal{T}$) of a piece in the r_j-division for each $j > i$.

We first show how to efficiently compute the external DDGs for all k-tuples of pieces of an r-division, $r \in \nabla$.

LEMMA 5.1. *Given $r_i \in \nabla$ and an integer $d \leq \frac{n}{r_i}$, one can compute DDG°_{ext} for all d-tuples of pieces of each r_t-division, $t \geq i$, in time $\mathcal{O}(\frac{(cn)^d}{r_i^{d-1}} \frac{1}{(d-2)!} \log^2 n)$ for some constant $c > 1$.*

Proof. We prove this lemma by induction on ∇ from top to bottom. For $r_\nu = n$, the only piece is G, and $DDG^\circ_{ext}(G)$ is the empty graph. Assume inductively that we have $DDG^\circ_{ext}(R_1, \dots, R_d)$ for every d-tuple $(R_1, \dots, R_d)$ of pieces at the r_{i+1}-division. Let $Q_1, \dots, Q_d$ be pieces at the r_i-division. Note that every piece at level r_i is contained in some piece at level r_{i+1}, but a piece at level r_{i+1} might contain multiple pieces at level r_i. Let $R_1, \dots, R_d$ be pieces of the r_{i+1}-division such that each Q_j is a subgraph of some $R_{j'}$. Let $\mathcal{Q}_{R_j}$ be the maximal subset of $\{Q_1, \dots, Q_d\}$ such that each piece in $\mathcal{Q}_{R_j}$ is contained in R_j. For every $j \in \{1, \dots d\}$ let $R'_j = R_j \setminus (\bigcup \mathcal{Q}_{R_j})$ (i.e. the allowed internal part of R_j). Since R_j and each $Q_m \in \mathcal{Q}_{R_j}$ have $\mathcal{O}(\sqrt{r_{i+1}})$ and $\mathcal{O}(\sqrt{r_i})$ boundary vertices respectively, R'_j has $\mathcal{O}(\sqrt{r_{i+1}} + \sqrt{r_i}|\mathcal{Q}_{R_j}|) = \mathcal{O}(|\mathcal{Q}_{R_j}|\sqrt{r_{i+1}})$ boundary vertices (recall that $r_{i+1}/r_i = b$).

We compute $DDG^\circ_{R'_j}$ in a similar manner to the query of Section 3 by running FR-Dijkstra on the union of the following DDG°s. For each piece $Q_m \in \mathcal{Q}_{R_j}$, for each ancestor Q of Q_m (including Q_m) that is a strict descendant of R_j in $\mathcal{T}$, we take the DDG°_P of the sibling P of Q if P contains no piece of $\mathcal{Q}_{R_j}$. The pieces of $\mathcal{Q}_{R_j}$ have $\mathcal{O}(|\mathcal{Q}_{R_j}|\sqrt{r_i})$ boundary vertices in total and the total number of boundary vertices for their ancestors is bounded by $\mathcal{O}(|\mathcal{Q}_{R_j}|\sqrt{r_{i+1}})$. Running FR-Dijkstra from each of the $\mathcal{O}(|\mathcal{Q}_{R_j}|\sqrt{r_{i+1}})$ boundary vertices of R'_j yields $DDG^\circ_{R'_j}$ and requires $\mathcal{O}(|\mathcal{Q}_{R_j}|\sqrt{r_{i+1}}|\mathcal{Q}_{R_j}|\sqrt{r_{i+1}} \log^2 n) = \mathcal{O}(|\mathcal{Q}_{R_j}|^2 r_{i+1} \log^2 n)$ time in total. When summing over $R_1, \dots, R_d$ we get $\sum_{j=1}^d |\mathcal{Q}_{R_j}|^2 r_{i+1} \log^2 n \leq r_{i+1} \log^2 n \big(\sum_{j=1}^d |\mathcal{Q}_{R_j}|\big)^2 = d^2 r_{i+1} \log^2 n$. The inequality is due to the Cauchy-Schwarz inequality and the equality follows from the fact that $\sum_{j=1}^d |\mathcal{Q}_{R_j}| = d$.

Let $\mathcal{D} = DDG^\circ_{ext}(R_1, \dots, R_d) \bigcup (\bigcup_{j=1}^{d} DDG^\circ_{R'_j})$. $DDG^\circ_{ext}(R_1, \dots, R_d)$ and $\bigcup_{j=1}^{d} DDG^\circ_{R'_j}$ contribute $\mathcal{O}(d\sqrt{r_{i+1}})$ and $\mathcal{O}(d(\sqrt{r_{i+1}} + \sqrt{r_i}))$ boundary vertices to $\mathcal{D}$, respectively. We run FR-Dijkstra on $\mathcal{D}$ from each boundary vertex of Q_m for $m \in \{1, \dots d\}$. There are $\mathcal{O}(d\sqrt{r_i})$ such boundary vertices, so this requires $\mathcal{O}(d\sqrt{r_i} d(\sqrt{r_{i+1}} + \sqrt{r_i}) \log^2 n) = \mathcal{O}(d^2 r_{i+1} \log^2 n)$ time, and yields $DDG^\circ_{ext}(Q_1, \dots, Q_d)$.

We can thus compute $DDG^\circ_{ext}(Q_1, \dots, Q_d)$ for all d-tuples at level r_i in $\mathcal{O}((g(n, r_i, d) d^2 r_{i+1} \log^2 n) = \mathcal{O}(\frac{(cn)^d}{r_i^d} r_{i+1} \frac{1}{d!} d^2 \log^2 n) = \mathcal{O}(\frac{(cn)^d}{r_i^{d-1}} \frac{1}{(d-2)!} \log^2 n)$ time, assuming that we have the DDG°_{ext}s for all d-tuples of pieces of r_t-divisions, $t > i$.

The time to compute the DDG°_{ext}s for all d-tuples of pieces of all r_t-divisions, $t > i$, is, inductively, $\mathcal{O}\big((cn)^d \frac{1}{(d-2)!} \log^2 n \sum_{t=i+1}^{\nu} \frac{1}{r_t^{d-1}}\big)$, and $\sum_{t=i+1}^{\nu} \frac{1}{r_t^{d-1}} = \frac{1}{r_i^{d-1}} \sum_{t=1}^{\nu-i} (\frac{1}{b^{d-1}})^t = \mathcal{O}(\frac{1}{r_i^{d-1}})$ since $b^{d-1} > 1$. Thus computing the DDG°_{ext}s for d-tuples of pieces of the r_i-division dominates the time complexity. ☐

We next show how to efficiently compute the additive distances with respect to which the Voronoi diagrams stored by our oracle are computed.

LEMMA 5.2. *Let $\mathcal{R}_r$ be an r-division, such that $r \in \nabla$, and let $d \leq \frac{n}{r}$ be an integer. For all d-tuples of pieces $R_1, \dots, R_d$ in $\mathcal{R}_r$ and for all pieces $Q \in \mathcal{T}$ such that Q does not contain any of the pieces R_i, and Q is a sibling of a node in the root to-R_i path in $\mathcal{T}$ for some R_i, one can compute the distances from each $y \in \bigcup_{i=1}^{d} \partial R_i$ to each boundary vertex of Q in the graph $G \setminus \big((\bigcup_{i=1}^{d} R_i) \setminus \{y\}\big)$ in time $\mathcal{O}\big(\frac{(cn)^d}{r^d} \frac{1}{(d-2)!} \sqrt{ndr} \log^3 n\big)$ in total, for some constant $c > 1$.*

Proof. Let us consider a d-tuple of pieces $(R_1, \dots, R_d)$ and a piece Q, satisfying the properties in the statement of the lemma. To compute the desired distances, we run FR-Dijkstra from each $y \in \bigcup_{i=1}^{d} \partial R_i$ on the union of the following DDGs:

1. DDG°_Q.
2. For each piece $R_i \in \{R_1, \dots, R_d\}$ for each ancestor A of R_i (including R_i) in $\mathcal{T}$, we take the DDG°_B of the sibling B of A if B contains no piece of $R_1, \dots, R_d$.

This correctly computes the distances by the same arguments that were applied in Section 3. It remains to analyze the time complexity. Consider the (n/d)-division of G in $\mathcal{T}$. By the same argument that was applied in the proof of Lemma 3.2 we can bound the number of boundary vertices for all the included DDG°s by $\mathcal{O}(\sqrt{dn})$. There are $\mathcal{O}(d\sqrt{r})$ choices of $y \in \bigcup_{i=1}^{d} \partial R_i$, so the time required to run FR-Dijkstra from each y is $\mathcal{O}(d\sqrt{r}\sqrt{dn}\log^2 n) = \mathcal{O}(d\sqrt{nrd}\log^2 n)$.

Each piece $R_i \in \{R_1, \ldots, R_d\}$ has $\mathcal{O}(\log n)$ nodes in the root-to-R_i path in $\mathcal{T}$, hence computing the distances for all possible choices of Q requires time $\mathcal{O}(d^2\sqrt{nrd}\log^3 n)$. Finally, in order to compute the distances for all d-tuples of pieces we need time $\mathcal{O}((g(n,r,d)d^2\sqrt{nrd}\log^3 n) = \mathcal{O}((\frac{(cn)^d}{r^d})\frac{1}{d!}d^2\sqrt{nrd}\log^3 n) = \mathcal{O}((\frac{(cn)^d}{r^d})\frac{1}{(d-2)!}\sqrt{nrd}\log^3 n)$. ◻

LEMMA 5.3. *We can compute the representation of the Voronoi diagrams described in Section 2 with respect to sets of sites of total cardinality $\mathcal{S}$, each corresponding to a piece $P \in \mathcal{T}$ and consisting of nodes of ∂P that lie on a single hole of P, and specifying an additive weight for each of these nodes in time $\tilde{\mathcal{O}}(n^2 + \mathcal{S})$ in total.*

Proof. We apply Theorem 2.2 and construct all the Voronoi diagrams corresponding to each of the $\mathcal{O}(1)$ holes of each piece as a batch. For a hole h of a piece P, the time required is $\tilde{\mathcal{O}}(|P||\partial P|^2 + \sum_h \mathcal{S}_{P,h})$, where $\mathcal{S}_{P,h}$ is the total cardinality of the sets of sites corresponding to nodes of ∂P lying on h. Then we have that

$$\sum_{P\in\mathcal{T}} \left(|P||\partial P|^2 + \sum_h |\mathcal{S}_{P,h}|\right) = \mathcal{O}(n^2 + \mathcal{S}),$$

by Proposition 2.2 and hence the stated bound follows. ◻

6 Dynamic Distance Oracles can handle Vertex Deletions

In this section we briefly explain how the techniques of Section 3, and specifically our notion of strict dense distance graph DDG° can be used to facilitate vertex deletions in dynamic distance oracles for planar graphs. The dynamic distance oracle of [16] for non-negative edge-weight updates was improved and simplified in [24]. In [24], the algorithm obtains an r-division of G, and then computes and preprocesses the DDGs of the pieces of the r-division in $\mathcal{O}(n\log n)$ time to allow for FR-Dijkstra computations in the union of these DDGs in $\mathcal{O}(\frac{n}{\sqrt{r}}\log^2 n)$. For a given query asking for the distance from some vertex u to some vertex v, the algorithm performs standard Dijkstra computations within the piece containing u (resp. v) to compute the distances from u to the boundary vertices of the piece (resp. from the boundary vertices of the piece to v). The algorithm then combines this with an FR-Dijkstra computation on the boundary vertices of the r-division. Given an edge update, only the DDG of the unique piece in the r-division containing the updated edge needs to get updated, and this requires $\mathcal{O}(r\log r)$ time. The balance is at $r = n^{2/3}\log^{2/3} n$, yielding $\mathcal{O}(n^{2/3}\log^{5/3} n)$ time per update and query. This result was extended in [22], where the authors showed how to allow for edge insertions (not violating the planarity of the embedding) and edge deletions and further in [23] where the authors showed how to handle arbitrary (i.e. also negative) edge-weight updates. The time complexity was improved by a $\log^{4/3}\log n$ factor in [19].

We observe that, by using DDG°s instead of the standard DDGs, vertex deletions can also be handled as follows. Each vertex is either a boundary vertex in each piece of the r-division containing it, or an internal vertex in a unique piece. If a deleted vertex is a boundary vertex, we just mark it as such and do not relax edges outgoing from it during (FR-)Dijkstra computations. If a deleted vertex is internal, we recompute the DDG° of the piece containing it, and reprocess it in time $\mathcal{O}(r\log r)$ exactly as in the case of edge-weight updates. The only slightly technical issue we need to take into account is that in Section 3, edge weights in DDG° are shifted by the large constant C (recall that C is defined as twice the sum of edge weights in the entire graph G). The problem is that C might change after each update operation, and this update affects the weights of all the edges in all DDG°s. This can be easily solved using indirection. Instead of using the explicit value of C in each edge weight, we represent C symbolically, and store the actual value of C explicitly at some placeholder. Updating C can be done in constant time because only the explicit value at the placeholder needs to be updated. Whenever an edge weight is required by the algorithm, it is computed on the fly in constant time using the value of C stored in the placeholder. The data structures underlying FR-Dijkstra do not make use of any integer data structures like predecessor data structures — all used data structures are comparison based. Hence, since the value of C is greater than all edge-weights at the time they are built, they are identical to the data structures that would have been built for this piece with any subsequent value of C. Vertex additions do not alter shortest paths, and hence can be treated trivially. Note that, as in [22], we can afford to recompute the entire data structure from scratch after every $\mathcal{O}(\sqrt{r})$ operations. This guarantees that the number of vertices and number of boundary vertices in each piece remain $\mathcal{O}(r)$ and $\mathcal{O}(\sqrt{r})$, respectively, throughout. We formalize the above discussion in the following theorem.

THEOREM 6.1. *A planar graph G can be preprocessed in time $\mathcal{O}(n\frac{\log^2 n}{\log\log n})$ so that edge-weight updates, edge insertions not violating the planarity of the embedding, edge deletions, vertex insertions and deletions, and distance queries can be performed in time $\mathcal{O}(n^{2/3}\frac{\log^{5/3} n}{\log^{4/3}\log n})$ each, using $\mathcal{O}(n)$ space.*

7 Final Remarks

Perhaps the most intriguing open question related to our results is whether it is possible to answer distance queries subject to even one failure in time $\tilde{\mathcal{O}}(1)$ with an $o(n^2)$-size oracle. Recall that the best known exact failure-free distance oracle that answers queries in $\tilde{\mathcal{O}}(1)$ occupies $\tilde{\mathcal{O}}(n^{3/2})$ space [20].

References

[1] Amir Abboud and Søren Dahlgaard. Popular conjectures as a barrier for dynamic planar graph algorithms. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 477–486. IEEE Computer Society, 2016.

[2] Ittai Abraham, Shiri Chechik, and Cyril Gavoille. Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 1199–1218. ACM, 2012.

[3] Srinivasa Rao Arikati, Danny Z. Chen, L. Paul Chew, Gautam Das, Michiel H. M. Smid, and Christos D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In Josep Díaz and Maria J. Serna, editors, *Algorithms - ESA '96, Fourth Annual European Symposium, Barcelona, Spain, September 25-27, 1996, Proceedings*, volume 1136 of *Lecture Notes in Computer Science*, pages 514–528. Springer, 1996.

[4] Surender Baswana, Utkarsh Lath, and Anuradha S. Mehta. Single source distance oracle for planar digraphs avoiding a failed node or link. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 223–232, 2012.

[5] Aaron Bernstein and David R. Karger. A nearly optimal oracle for avoiding failed vertices and edges. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 101–110. ACM, 2009.

[6] Sergio Cabello. Many distances in planar graphs. *Algorithmica*, 62(1-2):361–381, 2012.

[7] Sergio Cabello, Erin W. Chambers, and Jeff Erickson. Multiple-source shortest paths in embedded graphs. *SIAM J. Comput.*, 42(4):1542–1571, 2013.

[8] Danny Z. Chen and Jinhui Xu. Shortest path queries in planar graphs. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 469–478. ACM, 2000.

[9] Vincent Cohen-Addad, Søren Dahlgaard, and Christian Wulff-Nilsen. Fast and compact exact distance oracle for planar graphs. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 962–973. IEEE Computer Society, 2017.

[10] Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008.

[11] Hristo Djidjev. On-line algorithms for shortest path problems on planar digraphs. In Fabrizio d'Amore, Paolo Giulio Franciosa, and Alberto Marchetti-Spaccamela, editors, *Graph-Theoretic Concepts in Computer Science, 22nd International Workshop, WG '96, Cadenabbia (Como), Italy, June 12-14, 1996, Proceedings*, volume 1197 of *Lecture Notes in Computer Science*, pages 151–165. Springer, 1996.

[12] Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 506–515. SIAM, 2009.

[13] Yuval Emek, David Peleg, and Liam Roditty. A near-linear-time algorithm for computing replacement paths in planar directed graphs. *ACM Trans. Algorithms*, 6(4):64:1–64:13, 2010.

[14] David Eppstein. Finding the k shortest paths. *SIAM J. Comput.*, 28(2):652–673, 1998.

[15] Jeff Erickson, Kyle Fox, and Luvsandondov Lkhamsuren. Holiest minimum-cost paths and flows in surface graphs. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1319–1332. ACM, 2018.

[16] Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *J. Comput. Syst. Sci.*, 72(5):868–889, 2006.

[17] Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987.

[18] Pawel Gawrychowski, Haim Kaplan, Shay Mozes, Micha Sharir, and Oren Weimann. Voronoi diagrams on planar graphs, and computing the diameter in deterministic $\tilde{O}(n^{5/3})$ time. In *Proceedings of the Twenty-*

Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018, pages 495–514, 2018.

[19] Pawel Gawrychowski and Adam Karczmarz. Improved bounds for shortest paths in dense distance graphs. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 61:1–61:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[20] Pawel Gawrychowski, Shay Mozes, Oren Weimann, and Christian Wulff-Nilsen. Better tradeoffs for exact distance oracles in planar graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 515–529, 2018.

[21] John Hershberger and Subhash Suri. Vickrey prices and shortest paths: What is an edge worth? In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 252–259. IEEE Computer Society, 2001.

[22] Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 313–322. ACM, 2011.

[23] Haim Kaplan, Shay Mozes, Yahav Nussbaum, and Micha Sharir. Submatrix maximum queries in monge matrices and partial monge matrices, and their applications. *ACM Trans. Algorithms*, 13(2):26:1–26:42, 2017.

[24] Philip N. Klein. Multiple-source shortest paths in planar graphs. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 146–155. SIAM, 2005.

[25] Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 505–514. ACM, 2013.

[26] Philip N. Klein, Shay Mozes, and Oren 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):30:1–30:18, 2010.

[27] Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. In Richard A. DeMillo, editor, *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1984, Washington, DC, USA*, pages 376–382. ACM, 1984.

[28] Gaspard Monge. *Mémoire sur la théorie des déblais et des remblais.* De l'Imprimerie Royale, 1781.

[29] Shay Mozes and Christian Sommer. Exact distance oracles for planar graphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 209–222, 2012.

[30] Shay Mozes and Christian Wulff-Nilsen. Shortest paths in planar graphs with real lengths in $O(n\mathrm{log}^2 n/\mathrm{loglog}n)$ time. In Mark de Berg and Ulrich Meyer, editors, *Algorithms - ESA 2010, 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010. Proceedings, Part II*, volume 6347 of *Lecture Notes in Computer Science*, pages 206–217. Springer, 2010.

[31] Noam Nisan and Amir Ronen. Algorithmic mechanism design (extended abstract). In Jeffrey Scott Vitter, Lawrence L. Larmore, and Frank Thomson Leighton, editors, *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 129–140. ACM, 1999.

[32] Oren Weimann and Raphael Yuster. Replacement paths and distance sensitivity oracles via fast matrix multiplication. *ACM Trans. Algorithms*, 9(2):14:1–14:13, 2013.

[33] Christian Wulff-Nilsen. Solving the replacement paths problem for planar directed graphs in o(n log n) time. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 756–765. SIAM, 2010.