Deterministic Combinatorial Replacement Paths and Distance Sensitivity Oracles

Noga Alon
Department of Mathematics, Princeton University, Princeton, NJ 08544, USA
Schools of Mathematics and Computer Science, Tel Aviv University, Tel Aviv 69978, Israel
nogaa@tau.ac.il

Shiri Chechik
Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel
shiri.chechik@gmail.com

Sarel Cohen
Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel
sarelcoh@post.tau.ac.il

Abstract

In this work we derandomize two central results in graph algorithms, replacement paths and distance sensitivity oracles (DSOs) matching in both cases the running time of the randomized algorithms.

For the replacement paths problem, let $G = (V, E)$ be a directed unweighted graph with n vertices and m edges and let P be a shortest path from s to t in G. The *replacement paths* problem is to find for every edge $e \in P$ the shortest path from s to t avoiding e. Roditty and Zwick [ICALP 2005] obtained a randomized algorithm with running time of $\widetilde{O}(m\sqrt{n})$. Here we provide the first deterministic algorithm for this problem, with the same $\widetilde{O}(m\sqrt{n})$ time. Due to matching conditional lower bounds of Williams et al. [FOCS 2010], our deterministic combinatorial algorithm for the replacement paths problem is optimal up to polylogarithmic factors (unless the long standing bound of $\widetilde{O}(mn)$ for the combinatorial boolean matrix multiplication can be improved). This also implies a deterministic algorithm for the second simple shortest path problem in $\widetilde{O}(m\sqrt{n})$ time, and a deterministic algorithm for the k-simple shortest paths problem in $\widetilde{O}(km\sqrt{n})$ time (for any integer constant $k > 0$).

For the problem of distance sensitivity oracles, let $G = (V, E)$ be a directed graph with real-edge weights. An f-Sensitivity Distance Oracle (f-DSO) gets as input the graph $G = (V, E)$ and a parameter f, preprocesses it into a data-structure, such that given a query (s, t, F) with $s, t \in V$ and $F \subseteq E \cup V, |F| \leq f$ being a set of at most f edges or vertices (failures), the query algorithm efficiently computes the distance from s to t in the graph $G \setminus F$ (i.e., the distance from s to t in the graph G after removing from it the failing edges and vertices F).

For weighted graphs with real edge weights, Weimann and Yuster [FOCS 2010] presented several randomized f-DSOs. In particular, they presented a combinatorial f-DSO with $\widetilde{O}(mn^{4-\alpha})$ preprocessing time and subquadratic $\widetilde{O}(n^{2-2(1-\alpha)/f})$ query time, giving a tradeoff between preprocessing and query time for every value of $0 < \alpha < 1$. We derandomize this result and present a combinatorial deterministic f-DSO with the same asymptotic preprocessing and query time.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Dynamic graph algorithms

Keywords and phrases replacement paths, distance sensitivity oracles, derandomization

Digital Object Identifier 10.4230/LIPIcs.ICALP.2019.12

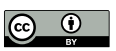
Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at [3], `https://arxiv.org/abs/1905.07483`.

Funding *Noga Alon*: Research supported in part by NSF grant DMS-1855464, ISF grant 281/17 and GIF grant G-1347-304.6/2016.

Shiri Chechik: Research supported in part by the Israel Science Foundation grant No. 1528/15 and the Blavatnik Fund.

Sarel Cohen: Research supported in part by the Israel Science Foundation grant No. 1528/15 and the Blavatnik Fund.



© Noga Alon, Shiri Chechik, and Sarel Cohen;
licensed under Creative Commons License CC-BY

46th International Colloquium on Automata, Languages, and Programming (ICALP 2019).
Editors: Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi;
Article No. 12; pp. 12:1–12:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In many algorithms used in computing environments such as massive storage devices, large scale parallel computation, and communication networks, recovering from failures must be an integral part. Therefore, designing algorithms and data structures whose running time is efficient even in the presence of failures is an important task. In this paper we study variants of shortest path queries in setting with failures.

The computation of shortest paths and distances in the presence of failures was extensively studied. Two central problems researched in this field are the Replacement Paths problem and Distance Sensitivity Oracles, we define these problems hereinafter.

The Replacement Paths problem. (See, e.g., [26, 28, 16, 14, 21, 27, 6, 30, 22, 23, 24, 25, 29, 15].) Let $G = (V, E)$ be a graph (directed or undirected, weighted or unweighted) with n vertices and m edges and let $P_G(s, t)$ be a shortest path from s to t. For every edge $e \in P_G(s, t)$ a replacement path $P_G(s, t, e)$ is a shortest path from s to t in the graph $G \setminus \{e\}$ (which is the graph G after removing the edge e). Let $d_G(s, t, e)$ be the length of the path $P_G(s, t, e)$. The replacement paths problem is as follows: given a shortest path $P_G(s, t)$ from s to t in G, compute $d_G(s, t, e)$ (or an approximation of it) for every $e \in P_G(s, t)$.

Distance Sensitivity Oracles. (See, e.g., [9, 17, 7, 8, 10, 11, 12, 13, 19].) An f-Sensitivity Distance Oracle (f-DSO) gets as input a graph $G = (V, E)$ and a parameter f, preprocesses it into a data-structure, such that given a query (s, t, F) with $s, t \in V$ and $F \subseteq E \cup V, |F| \leq f$ being a set of at most f edges or vertices (failures), the query algorithm efficiently computes (exactly or approximately) $d_G(s, t, F)$ which is the distance from s to t in the graph $G \setminus F$ (i.e., in the graph G after removing from it the failing edges and vertices F). Here we would like to optimize several parameters of the data-structure: minimize the size of the oracle, support many failures f, have efficient preprocessing and query algorithms, and if the output is an approximation of the distance then optimize the approximation-ratio.

An important line of research in the theory of computer science is derandomization. In many algorithms and data-structures there exists a gap between the best known randomized algorithms and the best known deterministic algorithms. There has been extensive research on closing the gaps between the best known randomized and deterministic algorithms in many problems or proving that no deterministic algorithm can perform as good as its randomized counterpart. There also has been a long line of work on developing derandomization techniques, in order to obtain deterministic versions of randomized algorithms (e.g., Chapter 16 in [2]).

In this paper we derandomize algorithms and data-structures for computing distances and shortest paths in the presence of failures. Many randomized algorithms for computing shortest paths and distances use variants of the following sampling lemma (see Lemma 1 in Roditty and Zwick [26]).

▶ **Lemma 1** (Lemma 1 in [26]). *Let $D_1, D_2, \ldots, D_q \subseteq V$ satisfy $|D_i| > L$ for $1 \leq i \leq q$ and $|V| = n$. If $R \subseteq V$ is a random subset obtained by selecting each vertex, independently, with probability $(c \ln n)/L$, for some $c > 0$, then with probability of at least $1 - q \cdot n^{-c}$ we have $D_i \cap R \neq \emptyset$ for every $1 \leq i \leq q$.*

Our derandomization step of Lemma 1 is very simple, as described in Section 1.3, we use the folklore greedy approach to prove the following lemma, which is a deterministic version of Lemma 1.

▶ **Lemma 2** (See also Section 1.3). *Let $D_1, D_2, \ldots, D_q \subseteq V$ satisfy $|D_i| > L$ for $1 \leq i \leq q$ and $|V| = n$. One can deterministically find in $\widetilde{O}(qL)$ time a set $R \subset V$ such that $|R| = \widetilde{O}(n/L)$ and $D_i \cap R \neq \emptyset$ for every $1 \leq i \leq q$.*

We emphasize that the use of Lemma 2 is very standard and is not our main contribution. The main technical challenge is how to efficiently and deterministically compute a small number of sets $D_1, D_2, \ldots, D_q \subseteq V$ so that the invocation of Lemma 2 is fast.

1.1 Derandomizing the Replacment Paths Algorithm of Roditty and Zwick [26]

We derandomize the algorithm of Roditty and Zwick [26] and obtain a near optimal deterministic algorithm for the replacement paths problem in directed unweighed graphs (a problem which was open for more than a decade since the randomized algorithm was published) as stated in the following theorem.

▶ **Theorem 3.** *There exists a deterministic algorithm for the replacement paths problem in unweighted directed graphs whose runtime is $\widetilde{O}(m\sqrt{n})$. This algorithm is near optimal assuming the conditional lower bound of combinatorial boolean matrix multiplication of [29].*

The term "combinatorial algorithms" is not well-defined, and it is often interpreted as non-Strassen-like algorithms [4], or more intuitively, algorithms that do not use any matrix multiplication tricks. Arguably, in practice, combinatorial algorithms are to some extent considered more efficient since the constants hidden in the matrix multiplication bounds are high. On the other hand, there has been research done to make fast matrix multiplication practical, *e.g.*, [18, 5].

Vassilevska Williams and Williams [29] proved a subcubic equivalence between $\sqrt{n}$ occurrences of the combinatorial replacement paths problem in unweighted directed graphs and the combinatorial boolean multiplication (BMM) problem. More precisely, they proved that there exists some fixed $\epsilon > 0$ such that the combinatorial replacement paths problem can be solved in $O(mn^{1/2-\epsilon})$ time if and only if there exists some fixed $\delta > 0$ such that the combinatorial boolean matrix multiplication (BMM) can be solved in subcubic $O(n^{3-\delta})$ time. Giving a subcubic combinatorial algorithm to the BMM problem, or proving that no such algorithm exists, is a long standing open problem. This implies that either both problems can be polynomially improved, or neither of them does. Hence, assuming the conditional lower bound of combinatorial BMM, our combinatorial $\widetilde{O}(m\sqrt{n})$ algorithm for the replacement paths problem in unweighted directed graphs is essentially optimal (up to $n^{o(1)}$ factors).

The replacement paths problem is related to the k simple shortest paths problem, where the goal is to find the k simple shortest paths between two vertices. Using known reductions from the replacement paths problem to the k simple shortest paths problem, we close this gap as the following Corollary states.

▶ **Corollary 4.** *There exists a deterministic algorithm for computing k simple shortest paths in unweighted directed graphs whose runtime is $\widetilde{O}(km\sqrt{n})$.*

The trivial $\widetilde{O}(mn)$ time algorithm for solving the replacement paths problem in directed weighted graphs (simply, for every edge $e \in P_G(s,t)$ run Dijkstra in the graph $G \setminus \{e\}$) is deterministic and near optimal (according to a conditional lower bound by [29]). To the best of our knowledge the only deterministic combinatorial algorithms known for directed unweighted graphs are the algorithms for general directed weighted graphs whose runtime is $\widetilde{O}(mn)$ leaving a significant gap between the randomized and deterministic algorithms. As mentioned above, in this paper we derandomize the $\widetilde{O}(m\sqrt{n})$ algorithm of Roditty and Zwick [26] and close this gap. More related work can be found in the full version.

1.2 Derandomizing the Combinatorial Distance Sensitivity Oracle of Weimann and Yuster [27]

Our second result is derandomizing the combinatorial distance sensitivity oracle of Weimann and Yuster [27] and obtaining the following theorem.

▶ **Theorem 5.** *Let $G = (V, E)$ be a directed graph with real edge weights, let $|V| = n$ and $|E| = m$. There exists a deterministic algorithm that given G and parameters $f = O(\frac{\log n}{\log \log n})$ and $0 < \alpha < 1$ constructs an f-sensitivity distance oracle in $\widetilde{O}(mn^{4-\alpha})$ time. Given a query (s, t, F) with $s, t \in V$ and $F \subseteq E \cup V, |F| \leq f$ being a set of at most f edges or vertices (failures), the deterministic query algorithm computes in $\widetilde{O}(n^{2-2(1-\alpha)/f})$ time the distance from s to t in the graph $G \setminus F$.*

We remark that while our focus in this paper is in computing distances, one may obtain the actual shortest path in time proportional to the number of edges of the shortest paths, using the same algorithm for obtaining the shortest paths in the replacement paths problem [26], and in the distance sensitivity oracles case [27].

1.3 Technical Contribution and Our Derandomization Framework

Let $\mathcal{A}$ be a random algorithm that uses Lemma 1 for sampling a subset of vertices $R \subseteq V$. We say that $\mathcal{P} = \{D_1, \ldots, D_q\}$ is a set of *critical* paths for the randomized algorithm $\mathcal{A}$ if $\mathcal{A}$ uses the sampling Lemma 1 and it is sufficient for the correctness of algorithm $\mathcal{A}$ that R is a hitting set for $\mathcal{P}$ (i.e., every path in $\mathcal{P}$ contains at least one vertex of R). According to Lemma 2 one can derandomize the random selection of the hitting set R in time that depends on the number of paths in $\mathcal{P}$. Therefore, in order to obtain an efficient derandomization procedure, we want to find a small set $\mathcal{P}$ of critical paths for the randomized algorithms.

Our main technical contribution is to show how to compute a small set of critical paths that is sufficient to be used as input for the greedy algorithm stated in Lemma 2.

Our framework for derandomizing algorithms and data-structures that use the sampling Lemma 1 is given in Figure 1.

1. **Step 1**: Prove the existence of a small set of critical paths $\{D_1, \ldots, D_q\}$ such that $|D_i| > L$ and show that it is sufficient for the correctness of the randomized algorithm that the set R obtained by Lemma 1 hits all the paths $D_1, \ldots, D_q$.
2. **Step 2**: Find an efficient algorithm to compute the paths $D_1, \ldots, D_q$.
3. **Step 3**: Use a deterministic algorithm to compute a small subset $R \subseteq V$ of vertices such that $D_i \cap R \neq \emptyset$ for every $1 \leq i \leq q$. For example, one can use the greedy algorithm of Lemma 2 or the blocker set algorithm of [20] to find a subset $R \subset V$ of $\widetilde{O}(n/L)$ vertices.

Figure 1 Our derandomization framework to derandomize algorithms that use the sampling Lemma 1.

Our first main technical contribution, denoted as Step 1 in Figure 1, is proving the existence of small sets of critical paths for the randomized replacement path algorithm of Roditty and Zwick [26] and for the distance sensitivity oracles of Weimann and Yuster [27]. Our second main technical contribution, denoted as Step 2 in Figure 1, is developing algorithms to efficiently compute these small sets of critical paths.

For the replacement paths problem, Roditty and Zwick [26] proved the existence of a critical set of $O(n^2)$ paths, each path containing at least $\lceil\sqrt{n}\rceil$ edges. Simply applying Lemma 2 on this set of paths requires $\widetilde{O}(n^{2.5})$ time which is too much, and it is also not clear from their algorithm how to efficiently compute this set of critical paths. As for Step 1, we prove the existence of a small set of $O(n)$ critical paths, each path contains $\lceil\sqrt{n}\rceil$ edges, and for Step 2, we develop an efficient algorithm that computes this set of critical paths in $\widetilde{O}(m\sqrt{n})$ time.

For the problem of distance sensitivity oracles, Weimann and Yuster [27] proved the existence of a critical set of $O(n^{2f+3})$ paths, each path containing $n^{(1-\alpha)/f}$ edges (where $0 < \alpha < 1$). Simply applying Lemma 2 on this set of paths requires $\widetilde{O}(n^{2f+3+(1-\alpha)/f})$ time which is too much, and here too, it is also not clear from their algorithm how to efficiently and deterministically compute this set of critical paths. As for Step 1, we prove the existence of a small set of $O(n^{2+\epsilon})$ critical paths, each path contains $n^{(1-\alpha)/f}$ edges, and for Step 2, we develop an efficient deterministic algorithm that computes this set of critical paths in $\widetilde{O}(mn^{1+\epsilon})$ time.

For Step 3, we use the folklore greedy deterministic algorithm denoted here by GreedyPivotsSelection($\{D_1, \ldots, D_q\}$). Given as input the paths $D_1, \ldots, D_q$, each path contains at least L vertices, the algorithm chooses a set of pivots $R \subseteq V$ such that for every $1 \leq i \leq q$ it holds that $D_i \cap R \neq \emptyset$. In addition, it holds that $|R| = \widetilde{O}(\frac{n}{L})$ and the runtime of the algorithm is $\widetilde{O}(qL)$.

The GreedyPivotsSelection algorithm works as follows. Let $\mathcal{P} = \{D_1, \ldots, D_q\}$. Starting with $R \leftarrow \emptyset$, find a vertex $v \in V$ which is contained in the maximum number of sets of $\mathcal{P}$, add it to R and remove all the sets that contain v from $\mathcal{P}$. Repeat this process until $\mathcal{P} = \emptyset$.

The following greedy selection lemma is folklore and we prove it in the full version.

▶ **Lemma 6.** *Let $1 \leq L \leq n$ and $1 \leq q < poly(n)$ be two integers. Let $D_1, \ldots, D_q \subseteq V$ be paths satisfying $|D_i| \geq L$ for every $1 \leq i \leq q$. The algorithm GreedyPivotsSelection($\{D_1, \ldots, D_q\}$) finds in $\widetilde{O}(qL)$ time a set $R \subset V$ such that for every $1 \leq i \leq q$ it holds that $R \cap D_i \neq \emptyset$ and $|R| = O(\frac{n \log q}{L}) = \widetilde{O}(n/L)$.*

Related Work - the Blocker Set Algorithm of King. We remark that the GreedyPivotsSelection algorithm is similar to the blocker set algorithm described in [20] for finding a hitting set for a set of paths. The blocker set algorithm was used in [20] to develop sequential dynamic algorithms for the APSP problem. Additional related work is that of Agarwal et al. [1]. They presented a deterministic distributed algorithm to compute APSP in an edge-weighted directed or undirected graph in $\widetilde{O}(n^{3/2})$ rounds in the Congest model by incorporating a deterministic distributed version of the blocker set algorithm.

While our derandomization framework uses the greedy algorithm (or the blocker set algorithm) to find a hitting set of vertices for a critical set of paths $D_1, \ldots, D_q$, we stress that our main contribution are the techniques to reduce the number of sets q the greedy algorithm must hit (Step 1), and the algorithms to efficiently compute the sets $D_1, \ldots, D_q$ (Step 2). These techniques are our main contribution, which enable us to use the greedy algorithm (or the blocker set algorithm) for a wider range of problems. Specifically, these techniques allow us to derandomize the best known random algorithms for the replacement paths problem and distance sensitivity oracles. We believe that our techniques can also be leveraged for additional related problems which use a sampling lemma similar to Lemma 1.

Outline. The structure of the paper is as follows. In Section 1.4 we describe some preliminaries and notations. In Section 2 we apply our framework to the replacement paths algorithm of Roditty and Zwick [26]. In Section 3 we apply our framework to the DSO of Weimann and Yuster for graphs with real-edge weights [27].

1.4 Preliminaries

Let $G = (V, E)$ be a directed weighted graph with n vertices and m edges with real edge weights $\omega(\cdot)$. Given a path P in G we define its weight $\omega(P) = \Sigma_{e \in E(P)} \omega(e)$.

Given $s, t \in V$, let $P_G(s,t)$ be a shortest path from s to t in G and let $d_G(s,t) = \omega(P_G(s,t))$ be its length, which is the sum of its edge weights. Let $|P_G(s,t)|$ denote the number of edges along $P_G(s,t)$. Note that for unweighted graphs we have $|P_G(s,t)| = d_G(s,t)$. When G is known from the context we sometimes abbreviate $P_G(s,t), d_G(s,t)$ with $P(s,t), d(s,t)$ respectively.

We define the path concatenation operator $\circ$ as follows. Let $P_1 = (x_1, x_2, \ldots, x_r)$ and $P_2 = (y_1, y_2, \ldots, y_t)$ be two paths. Then $P = P_1 \circ P_2$ is defined as the path $P = (x_1, x_2, \ldots, x_r, y_1, y_2, \ldots, y_t)$, and it is well defined if either $x_r = y_1$ or $(x_r, y_1) \in E$.

For a graph H we denote by $V(H)$ the set of its vertices, and by $E(H)$ the set of its edges. When it is clear from the context, we abbreviate $e \in E(H)$ by $e \in H$ and $v \in V(H)$ by $v \in H$.

Let P be a path which contains the vertices $u, v \in V(P)$ such that u appears before v along P. We denote by $P[u..v]$ the subpath of P from u to v.

For every edge $e \in P_G(s,t)$ a replacement path $P_G(s,t,e)$ for the triple (s,t,e) is a shortest path from s to t avoiding e. Let $d_G(s,t,e) = \omega(P_G(s,t,e))$ be the length of the replacement path $P_G(s,t,e)$.

We will assume, without loss of generality, that every replacement path $P_G(s,t,e)$ can be decomposed into a common prefix $\text{CommonPref}_{s,t,e}$ with the shortest path $P_G(s,t)$, a detour $\text{Detour}_{s,t,e}$ which is disjoint from the shortest path $P_G(s,t)$ (except for its first vertex and last vertex), and finally a common suffix $\text{CommonSuff}_{s,t,e}$ which is common with the shortest path $P_G(s,t)$. Therefore, for every edge $e \in P_G(s,t)$ it holds that $P_G(s,t,e) = \text{CommonPref}_{s,t,e} \circ \text{Detour}_{s,t,e} \circ \text{CommonSuff}_{s,t,e}$ (the prefix and/or suffix may be empty).

Let $F \subseteq V \cup E$ be a set of vertices and edges. We define the graph $G \setminus F = (V \setminus F, E \setminus F)$ as the graph obtained from G by removing the vertices and edges F. We define a replacement path $P_G(s,t,F)$ as a shortest path from s to t in the graph $G \setminus F$, and let $d_G(s,t,F) = w(P_G(s,t,e))$ be its length.

2 Deterministic Replacement Paths in $\widetilde{O}(m\sqrt{n})$ Time

In this section we apply our framework from Section 1.3 to the replacement paths algorithm of Roditty and Zwick [26].

The randomized algorithm by Roddity and Zwick as described in [26] takes $\widetilde{O}(m\sqrt{n})$ expected time. They handle separately the case that a replacement path has a short detour containing at most $\lceil \sqrt{n} \rceil$ edges, and the case that a replacement path has a long detour containing more than $\lceil \sqrt{n} \rceil$ edges. The first case is solved deterministically. The second case is solved by first sampling a subset of vertices R according to Lemma 1, where each vertex is sampled uniformly independently at random with probability $c \ln n / \sqrt{n}$ for large enough constant $c > 0$. Using this uniform sampling, it holds with high probability (of at least $1 - n^{-c+2}$) that for every long triple (s,t,e) (as defined hereinafter), the detour $\text{Detour}_{s,t,e}$ of the replacement path $P_G(s,t,e)$ contains at least one vertex of R.

▶ **Definition 7.** *Let $s,t \in V, e \in P_G(s,t)$. The triple (s,t,e) is a long triple if every replacement path from s to t avoiding e has its detour part containing more than $\lceil\sqrt{n}\rceil$ edges.*

Note that in Definition 7 we defined (s,t,e) to be a long triple if **every** replacement path from s to t avoiding e has a long detour (containing more than $\lceil\sqrt{n}\rceil$ edges). We could have defined (s,t,e) to be a long triple even if at least one replacement path from s to t avoiding e has a long detour (perhaps more similar to the definitions in [26]), however we find Definition 7 more convenient for the following reason. If (s,t,e) has a replacement path whose detour part contains at most $\lceil\sqrt{n}\rceil$ edges, then the algorithm of [26] for handling short detours finds deterministically a replacement path for (s,t,e). Hence, we only need to find the replacement paths for triples (s,t,e) for which every replacement path from s to t avoiding e has a long detour, and this is the case for which we define (s,t,e) as a long triple.

It is sufficient for the correctness of the replacement paths algorithm that the following condition holds; For every long triple (s,t,e) the detour $\text{Detour}_{s,t,e}$ of the replacement path $P_G(s,t,e)$ contains at least one vertex of R. As the authors of [26] write, the choice of the random set R is the only randomization used in their algorithm. To obtain a deterministic algorithm for the replacement paths problem and to prove Theorem 3, we prove the following deterministic alternative of Lemma 2.

▶ **Lemma 8** (Our derandomized version of Lemma 2 for the replacement paths algorithm)**.** *There exists an $\widetilde{O}(m\sqrt{n})$ time deterministic algorithm that computes a set $R \subseteq V$ of $\widetilde{O}(\sqrt{n})$ vertices, such that for every long triple (s,t,e) there exists a replacement path $P_G(s,t,e)$ whose detour part contains at least one of the vertices of R.*

Following the above description, in order to prove Theorem 3, that there exists an $\widetilde{O}(m\sqrt{n})$ deterministic replacement paths algorithm, it is sufficient to prove the derandomization Lemma 8, we do so in the following sections.

2.1 Step 1: the Method of Reusing Common Subpaths - Defining the Set $\mathcal{D}_n$

In this section we prove the following lemma.

▶ **Lemma 9.** *There exists a set $\mathcal{D}_n$ of at most n paths, each path of length exactly $\lceil\sqrt{n}\rceil$ with the following property; for every long triple (s,t,e) there exists a path $D \in \mathcal{D}_n$ and a replacement path $P_G(s,t,e)$ such that D is contained in the detour part of $P_G(s,t,e)$.*

In order to define the set of paths $\mathcal{D}_n$ and prove Lemma 9 we need the following definitions. Let $G' = G \setminus E(P_G(s,t))$ be the graph obtained by removing the edges of the path $P_G(s,t)$ from G. For two vertices u and v, let $d_{G'}(u,v)$ be the distance from u to v in G'.

We use the following definitions of the index $\rho(x)$, the set of vertices $V_{\sqrt{n}}$ and the set of paths $\mathcal{D}_n$.

▶ **Definition 10** (The index $\rho(x)$)**.** *Let $P_G(s,t) =< v_0,\ldots,v_k >$ and let $X = \{x \in V \mid \exists_{0\le i\le k}\ d_{G'}(v_i,x) = \lceil\sqrt{n}\rceil\}$ be the subset of all the vertices $x \in V$ such that there exists at least one index $0 \le i \le k$ with $d_{G'}(v_i,x) = \lceil\sqrt{n}\rceil$.*

For every vertex $x \in X$ we define the index $0 \le \rho(x) \le k$ to be the minimum index such that $d_{G'}(v_{\rho(x)},x) = \lceil\sqrt{n}\rceil$.

▶ **Definition 11** (The set of vertices $V_{\sqrt{n}}$)**.** *We define the set of vertices $V_{\sqrt{n}} = \{x \in X | \forall_{i<\rho(x)} d_{G'}(v_i,x) > \lceil\sqrt{n}\rceil\}$. In other words, $V_{\sqrt{n}}$ is the set of all vertices $x \in X$ such that for all the vertices v_i before $v_{\rho(x)}$ along $P_G(s,t)$ it holds that $d_{G'}(v_i,x) > \lceil\sqrt{n}\rceil$.*

▶ **Definition 12** (A set of paths $\mathcal{D}_n$). *For every vertex $x \in V_{\sqrt{n}}$, let $D(x)$ be an arbitrary shortest path from $v_{\rho(x)}$ to x in G' (whose length is $\lceil\sqrt{n}\rceil$ as $d_{G'}(v_{\rho(x)}, x) = \lceil\sqrt{n}\rceil$). We define $\mathcal{D}_n = \{D(x) | x \in V_{\sqrt{n}}\}$.*

Note that while $V_{\sqrt{n}}$ is uniquely defined (as it is defined according to distances between vertices) the set of paths $\mathcal{D}_n$ is not unique, as there may be many shortest paths from $v_{\rho(x)}$ to x in G', and we take $D(x) = P_{G'}(v_{\rho(x)}, x)$ to be an arbitrary such shortest path.

The basic intuition for the method of reusing common subpaths is as follows. Let $P_G(s,t,e_1), \ldots, P_G(s,t,e_r)$ be arbitrary replacement paths such that x is the $(\lceil\sqrt{n}\rceil + 1)^{\text{th}}$ vertex along the detours of all the replacement path $P_G(s,t,e_1), \ldots, P_G(s,t,e_r)$. Then one can construct replacement paths $P'_G(s,t,e_1), \ldots, P'_G(s,t,e_r)$ such that the subpath $D(x) \in \mathcal{D}_n$ is contained in all these replacement paths. Therefore, the subpath $D(x)$ is reused as a common subpath in many replacement paths. We utilize this observation in the following proof of Lemma 9.

Proof of Lemma 9. Obviously, the set $\mathcal{D}_n$ described in Definition 12 contains at most n paths, each path is of length exactly $\lceil\sqrt{n}\rceil$.

We prove that for every long triple (s,t,e) there exists a path $D \in \mathcal{D}_n$ and a replacement path $P'(s,t,e)$ *s.t.* D is contained in the detour part of $P'(s,t,e)$.

Let $P_G(s,t,e)$ be a replacement path for (s,t,e). Since (s,t,e) is a long triple then the detour part $\text{Detour}_{s,t,e}$ of $P_G(s,t,e)$ contains more than $\lceil\sqrt{n}\rceil$ edges. Let $x \in \text{Detour}_{s,t,e}$ be the $(\lceil\sqrt{n}\rceil + 1)^{\text{th}}$ vertex along $\text{Detour}_{s,t,e}$, and let v_j be the first vertex of $\text{Detour}_{s,t,e}$. Let P_1 be the subpath of $\text{Detour}_{s,t,e}$ from v_j to x and let P_2 be the subpath of $P_G(s,t,e)$ from x to t. In other words, $P_G(s,t,e) =< v_0, \ldots, v_j > \circ P_1 \circ P_2$. Since $\text{Detour}_{s,t,e}$ contains more than $\lceil\sqrt{n}\rceil$ edges and is disjoint from $P_G(s,t)$ except for the first and last vertices of $\text{Detour}_{s,t,e}$ and $P_1 \subset \text{Detour}_{s,t,e}$ it follows that P_1 is disjoint from $P_G(s,t)$ (except for the vertex v_j). In particular, since P_1 is a shortest path in $G \setminus \{e\}$ that is edge-disjoint from $P_G(s,t)$, then P_1 is also a shortest path in $G' = G \setminus E(P_G(s,t))$. We get that $d_{G'}(v_j, x) = |P_1| = \lceil\sqrt{n}\rceil$.

We prove that $j = \rho(x)$ and $x \in V_{\sqrt{n}}$. As we have already proved that $d_{G'}(v_j, x) = \lceil\sqrt{n}\rceil$, we need to prove that for every $0 \leq i < j$ it holds that $d_{G'}(v_i, x) > \lceil\sqrt{n}\rceil$. Assume by contradiction that there exists an index $0 \leq i < j$ such that $d_{G'}(v_i, x) \leq \lceil\sqrt{n}\rceil$. Then the path $\hat{P} =< v_0, \ldots, v_i > \circ P_{G'}(v_i, x) \circ P_2$ is a path from s to t that avoids e and its length is:

$$\begin{aligned} |\hat{P}| &= |< v_0, \ldots, v_i > \circ P_{G'}(v_i, x) \circ P_2| \\ &\leq i + \lceil\sqrt{n}\rceil + |P_2| \\ &< j + \lceil\sqrt{n}\rceil + |P_2| \\ &= |P_G(s, v_j) \circ P_1 \circ P_2| \\ &= |P_G(s,t,e)| \end{aligned}$$

This means that the path $\hat{P}$ is a path from s to t in $G \setminus \{e\}$ and its length is shorter than the length of the shortest path $P_G(s,t,e)$ from s to t in $G \setminus \{e\}$, which is a contradiction. We get that $d_{G'}(v_j, x) = \lceil\sqrt{n}\rceil$ and for every $0 \leq i < j$ it holds that $d_{G'}(v_i, x) > \lceil\sqrt{n}\rceil$. Therefore, according to Definitions 10 and 11 it holds that $j = \rho(x)$ and $x \in V_{\sqrt{n}}$.

Let $D(x) \in \mathcal{D}_n$, then according to Definition 12, $D(x)$ is a shortest path from $v_{\rho(x)}$ to x in G'. We define the path $P'(s,t,e) =< v_0, \ldots, v_{\rho(x)} > \circ D(x) \circ P_2$. It follows that $P'(s,t,e)$ is a path from s to t that avoids e and $|P'(s,t,e)| = |< v_0, \ldots, v_{\rho(x)} > \circ D(x) \circ P_2| = \rho(x) + \lceil\sqrt{n}\rceil + |P_2| = |P_G(s,t,e)| = d_G(s,t,e)$. Hence, $P'(s,t,e)$ is a replacement path for (s,t,e) such that $D(x) \subset P'(s,t,e)$ so the lemma follows. ◀

2.2 Step 2: the Method of Decremental Distances from a Path - Computing the Set $\mathcal{D}_n$

In this section we describe a decremental algorithm that enables us to compute the set of paths $\mathcal{D}_n$ in $\widetilde{O}(m\sqrt{n})$ time, proving the following lemma.

▶ **Lemma 13.** *There exists a deterministic algorithm for computing the set of paths $\mathcal{D}_n$ in $\widetilde{O}(m\sqrt{n})$ time.*

Our algorithm for computing the set of path $\mathcal{D}_n$ is a variant of the decremental SSSP (single source shortest paths) algorithm of King [20]. Our variant of the algorithm is used to find distances of vertices from a path rather than from a single source vertex as we define below.

Overview of the Deterministic Algorithm for Computing $\mathcal{D}_n$ in $\widetilde{O}(m\sqrt{n})$ Time. In the following description let $P = P_G(s,t)$. Consider the following assignment of weights ω to edges of G. We assign weight ϵ for every edge e on the path P, and weight 1 for all the other edges where ϵ is a small number such that $0 < \epsilon < 1/n$. We define a graph $G^w = (G, w)$ as the weighted graph G with edge weights ω. We define for every $0 \leq i \leq k$ the graph $G_i = G \setminus \{v_{i+1}, \ldots, v_k\}$ and the path $P_i = P \setminus \{v_{i+1}, \ldots, v_k\}$. We define the graph $G_i^w = (G_i, w)$ as the weighted graph G_i with edge weights ω.

The algorithm computes the graph G^w by simply taking G and setting all edge weights of $P_G(s,t)$ to be ϵ (for some small ϵ such that $\epsilon < 1/n$) and all other edge weights to be 1. The algorithm then removes the vertices of $P_G(s,t)$ from G^w one after the other (starting from the vertex that is closest to t). Loosely speaking after each vertex is removed, the algorithm computes the distances from s in the current graph. In each such iteration, the algorithm adds to $V^w_{\sqrt{n}}$ all vertices such that their distance from s in the current graph is between $\lceil\sqrt{n}\rceil$ and $\lceil\sqrt{n}\rceil + 1$. We will later show that at the end of the algorithm we have $V^w_{\sqrt{n}} = V_{\sqrt{n}}$. Unfortunately, we cannot afford running Dijkstra after the removal of every vertex of $P_G(s,t)$ as there might be n vertices on $P_G(s,t)$. To overcome this issue, the algorithm only maintains nodes at distance at most $\lceil\sqrt{n}\rceil + 1$ from s. In addition, we observe that to compute the SSSP from s in the graph after the removal of a vertex v_i we only need to spend time on nodes such that their shortest path from s uses the removed vertex. Roughly speaking, for these nodes we show that their distance from s rounded down to the closest integer must increase by at least 1 as a result of the removal of the vertex. Hence, for every node we spend time on it in at most $\lceil\sqrt{n}\rceil + 1$ iterations until its distance from s is bigger than $\lceil\sqrt{n}\rceil + 1$. As we will show later this will yield our desired running time.

In the full version we analyse the algorithm and prove Lemma 13.

Proof of Theorem 3. We summarize the $\widetilde{O}(m\sqrt{n})$ deterministic replacement paths algorithm and outline the proof of Theorem 3. First, compute in $\widetilde{O}(m\sqrt{n})$ time the set of paths $\mathcal{D}_n$ as in Lemma 13. Given $\mathcal{D}_n$, the deterministic greedy selection algorithm GreedyPivotsSelection($\mathcal{D}_n$) (as described in Lemma 2) computes a set $R \subset V$ of $\widetilde{O}(\sqrt{n})$ vertices in $\widetilde{O}(n\sqrt{n})$ time with the following property; every path $D \in \mathcal{D}_n$ contains at least one of the vertices of R. Theorem 3 follows from Lemmas 8, 9 and 13.

3 Deterministic Distance Sensitivity Oracles

Let $0 < \epsilon < 1$ and $1 \leq f = O(\frac{\log n}{\log\log n})$ be two parameters. In [27], Weimann and Yuster considered the following notion of intervals (note that in [27] they use a parameter $0 < \alpha < 1$ and we use a parameter $0 < \epsilon < 1$ such that $\epsilon = 1 - \alpha$). They define an interval of a long

simple path P as a subpath of P consisting of $n^{\epsilon/f}$ consecutive vertices, so every simple path induces less than n (overlapping) intervals. For every subset $F \subset E$ of at most f edges, and for every pair of vertices $u, v \in V$, let $P_G(u, v, F)$ be a shortest path from u to v in $G \setminus F$. The path $P_G(u, v, F)$ induces less than n (overlapping) intervals. The total number of possible intervals is less than $O(n^{2f+3})$ as each one of the (at most) $O(n^{2f+2})$ possible queries (u, v, F) corresponds to a shortest path $P_G(u, v, F)$ that induces less than n intervals.

▶ **Definition 14.** *Let $\mathcal{D}_f$ be defined as all the intervals (subpaths containing $n^{\epsilon/f}$ edges) of all the replacement paths $P_G(s, t, F)$ for every $s, t \in V, F \subseteq E \cup V$ with $|F| \leq f$.*

Weimann and Yuster apply Lemma 1 to find a set $R \subseteq V$ of $\widetilde{O}(n^{1-\epsilon/f})$ vertices that hit w.h.p. all the intervals $\mathcal{D}_f$. According to these bounds (that $\mathcal{D}_f$ contains $O(n^{2f+3})$ paths, each containing exactly $n^{\epsilon/f}$ edges) applying the greedy algorithm to obtain the set R deterministically according to Lemma 2 takes $\widetilde{O}(n^{2f+3+\epsilon/f})$ time, which is very inefficient.

In this section we assume that all weights are non-negative (so we can run Dijkstra's algorithm) and that shortest paths are unique, we justify these assumptions in the full version.

3.1 Step 1: the Method of Using Fault-Tolerant Trees to Significantly Reduce the Number of Intervals

In Lemma 15 we prove that the set of intervals $\mathcal{D}_f$ actually contains at most $O(n^{2+\epsilon})$ unique intervals, rather than the $O(n^{2f+3})$ naive upper bound mentioned above. From Lemmas 15 and 2 it follows that the GreedyPivotsSelection($\mathcal{D}_f$) finds in $\widetilde{O}(n^{2+\epsilon+\epsilon/f})$ time the subset $R \subseteq V$ of $\widetilde{O}(n^{1-\epsilon/f})$ vertices that hit all the intervals $\mathcal{D}_f$. In the full version we further reduce the time it takes for the greedy algorithm to compute the set of pivots R to $\widetilde{O}(n^{2+\epsilon})$.

▶ **Lemma 15.** $|\mathcal{D}_f| = O(n^{2+\epsilon})$.

In order to prove Lemma 15 we describe the fault-tolerant trees data-structure, which is a variant of the trees which appear in Appendix A of [9].

▶ **Definition 16.** *Let $P^L_G(s, t, F)$ be the shortest among the s-to-t paths in $G \setminus F$ that contain at most L edges and let $d^L_G(s, t, F) = \omega(P^L_G(s, t, F))$. In other words, $d^L_G(s, t, F) = \min\{\omega(P) \mid P \text{ is an } s-to-t \text{ path on at most } L \text{ edges}\}$. If there is no path from s to t in $G \setminus F$ containing at most L edges then we define $P^L_G(s, t, F) = \emptyset$ and $d^L_G(s, t, F) = \infty$. For $F = \emptyset$ we abbreviate $P^L_G(s, t, \emptyset) = P^L_G(s, t)$ as the shortest path from s to t that contains at most L edges, and $d^L_G(s, t) = d^L_G(s, t, \emptyset)$ as its length.*

Let $s, t \in V$ be vertices and let $L, f \geq 1$ be fixed integer parameters, we define the trees $FT^{L,f}(s, t)$ as follows.

- In the root of $FT^{L,f}(s, t)$ we store the path $P^L_G(s, t)$ (and its length $d^L_G(s, t)$), and also store the vertices and edges of $P^L_G(s, t)$ in a binary search tree $BST^L(s, t)$; If $P^L_G(s, t) = \emptyset$ then we terminate the construction of $FT^{L,f}(s, t)$.
- For every edge or vertex a_1 of $P^L_G(s, t)$ we recursively build a subtree $FT^{L,f}(s, t, a_1)$ as follows. Let $P^L_G(s, t, \{a_1\})$ be the shortest path from s to t that contains at most L edges in the graph $G \setminus \{a_1\}$. Then in the subtree $FT^{L,f}(s, t, a_1)$ we store the path $P^L_G(s, t, \{a_1\})$ (and its length $d^L_G(s, t, \{a_1\})$) and we also store the vertices and edges of $P^L_G(s, t, \{a_1\})$ in a binary search tree $BST^L(s, t, a_1)$; If $P^L_G(s, t, \{a_1\}) = \emptyset$ we terminate the construction of $FT^{L,f}(s, t, a_1)$. If $f > 1$ then for every vertex or edge a_2 in $P^L_G(s, t, \{a_1\})$ we recursively build the subtree $FT^{L,f}(s, t, a_1, a_2)$ as follows.

- For the recursive step, assume we want to construct the subtree $FT^{L,f}(s,t,a_1,\ldots,a_i)$. In the root of $FT^{L,f}(s,t,a_1,\ldots,a_i)$ we store the path $P^L_G(s,t,\{a_1,\ldots,a_i\})$ (and its length $d^L_G(s,t,\{a_1,\ldots,a_i\})$) and we also store the vertices and edges of $P^L_G(s,t,\{a_1,\ldots,a_i\})$ in a binary search tree $BST^L(s,t,a_1,\ldots,a_i)$. If $P^L_G(s,t,\{a_1,\ldots,a_i\})=\emptyset$ then we terminate the construction of $FT^{L,f}(s,t,a_1,\ldots,a_i)$. If $i<f$ then for every vertex or edge a_{i+1} in $P^L_G(s,t,\{a_1,\ldots,a_i\}))$ we recursively build the subtree $FT^{L,f}(s,t,a_1,\ldots,a_i,a_{i+1})$.

Observe that there are two conditions in which we terminate the recursive construction of $FT^{L,f}(s,t,a_1,\ldots,a_i)$:

- Either $i=f$ in which case $FT^{L,f}(s,t,a_1,\ldots,a_f)$ is a leaf node of $FT^{L,f}(s,t)$ and we store in the leaf node $FT^{L,f}(s,t,a_1,\ldots,a_f)$ the path $P^L_G(s,t,\{a_1,\ldots,a_f\})$.
- Or there is no path from s to t in $G\setminus\{a_1,\ldots,a_i\}$ that contains at most L edges and then $FT^{L,f}(s,t,a_1,\ldots,a_i)$ is a leaf vertex of $FT^{L,f}(s,v)$ and we store in it $P^L_G(s,t,\{a_1,\ldots,a_i\})=\emptyset$.

Querying the tree $FT^{L,f}(s,t)$. Given a query (s,t,F) such that $F\subset V\cup E$ with $|F|=f$ we would like to compute $d^L_G(s,t,F)$ using the tree $FT^{L,f}(s,t)$.

The query procedure is as follows. Let $P^L_G(s,t)$ be the path stored in the root of $FT^{L,f}(s,t)$ (if the root of $FT^{L,f}(s,t)$ contains $\emptyset$ then we output that $d^L_G(s,t,F)=\infty$). First we check if $P^L_G(s,t)\cap F=\emptyset$ by checking if any of the elements $a_1\in F$ appear in $BST^L(s,t)$ (which takes $O(\log L)$ time for each element $a_1\in F$). If $P^L_G(s,t)\cap F=\emptyset$ we output $d^L_G(s,t,F)=d^L_G(s,t)$ (as $P^L_G(s,t)$ does not contain any of the vertices or edges in F). Otherwise, let $a_1\in P^L_G(s,t)\cap F$.

We continue the search similarly in the subtree $FT^{L,f}(s,t,a_1)$ as follows. Let $P^L_G(s,t,\{a_1\})$ be the path stored in the root of $FT^{L,f}(s,t,a_1)$ (if the root of $FT^{L,f}(s,t,a_1)$ contains $\emptyset$ then we output that $d^L_G(s,t,F)=\infty$). First we check if $P^L_G(s,t,\{a_1\})\cap F=\emptyset$ by checking if any of the elements $a_2\in F$ appear in $BST^L(s,t,a_1)$ (which takes $O(\log L)$ time for each element $a_2\in F$). If $P^L_G(s,t,\{a_1\})\cap F=\emptyset$ we output $d^L_G(s,t,F)=d^L_G(s,t,\{a_1\})$ (as $P^L_G(s,t,\{a_1\})$ does not contain any of the vertices or edges in F). Otherwise, let $a_2\in P^L_G(s,t,\{a_1\})\cap F$. We continue the search similarly in the subtrees $FT^{L,f}(s,t,a_1,a_2)$, $FT^{L,f}(s,t,a_1,a_2,\ldots,a_i)$ until we either reach a leaf node which contains $\emptyset$ (and in this case we output that $d^L_G(s,t,F)=\infty$) or we find a path $P^L_G(s,t,\{a_1,\ldots,a_i\})$ such that $P^L_G(s,t,\{a_1,\ldots,a_i\})\cap F=\emptyset$ and then we output $d^L_G(s,t,F)=d^L_G(s,t,\{a_1,\ldots,a_i\})$.

In the full version we prove the following lemma.

▶ **Lemma 17.** *Given the tree $FT^{L,f}(s,t)$ and a set of failures $F\subset V\cup E$ with $|F|\le f$, the query procedure computes the distance $d^L_G(s,t,F)$ in $O(f^2\log L)$ time.*

We are now ready to prove lemma 15 asserting that $|\mathcal{D}_f|=O(n^{2+\epsilon})$.

Proof of Lemma 15. Let $L=n^{\epsilon/f}$ and let $\mathcal{D}$ be the set of all the unique shortest paths $P^L_G(s,t,\{a_1,\ldots,a_i\})$ stored in all the nodes of all the trees $\{FT^{L,f}(s,t)\}_{s,t\in V}$. Since the number of nodes in every tree $FT^{L,f}(s,t)$ is at most $L^f=(n^{\epsilon/f})^f=n^\epsilon$, and there are $O(n^2)$ trees (one tree for every pair of vertices $s,t\in V$) we get that the number of nodes in all the trees $\{FT^{L,f}(s,t)\}_{s,t\in V}$ is $O(n^{2+\epsilon})$ and hence $|\mathcal{D}|=O(n^{2+\epsilon})$.

We prove that $\mathcal{D}_f\subseteq\mathcal{D}$. By definition, $\mathcal{D}_f$ contains all the intervals (subpaths containing $n^{\epsilon/f}$ edges) of all the replacement paths $P_G(s,t,F)$ for every $s,t\in V, F\subseteq E\cup V$ with $|F|\le f$. Let $P\in\mathcal{D}_f$ be the unique shortest path, then P is a subpath containing $n^{\epsilon/f}$ edges of the replacement paths $P_G(s,t,F)$. Let u be the first vertex of P, and let v be the last vertex of P. Then P is a shortest path from u to v in $G\setminus F$, and since we assume

that the shortest paths our algorithms compute are unique then $P = P_G(u, v, F)$ is the unique shortest path from u to v in $G \setminus F$. Since P is assumed to be a path on exactly $L = n^{\epsilon/f}$ edges, then $P = P_G(u, v, F) = P_G^L(u, v, F)$. According to the query procedure in the tree $FT^{L,f}(u, v)$ and Lemma 17, if we query the tree $FT^{L,f}(u, v)$ with (u, v, F) then we reach a node $FT^{L,f}(u, v, a_1, \ldots, a_i)$ which contains the path $P_G^L(u, v, \{a_1, \ldots, a_i\})$ with $\{a_1, \ldots, a_i\} \subseteq F$ such that $P_G^L(u, v, \{a_1, \ldots, a_i\}) = P_G^L(u, v, F) = P$ is the shortest u-to-v path in $G \setminus F$. Hence, $P \in \mathcal{D}$ and thus $\mathcal{D}_f \subseteq \mathcal{D}$ and $|\mathcal{D}_f| \leq |\mathcal{D}| = O(n^{2+\epsilon})$ ◀

3.2 Step 2: Efficient Construction of the Fault-Tolerant Trees – Computing the Paths $\mathcal{D}_f$

Recall that we defined the trees $FT^{L,f}(u, v)$ with respect the parameters f (the maximum number of failures) and L (where we search for shortest paths among paths of at most L edges). The idea is to build the trees $FT^{L,f}(u, v)$ using dynamic programming having the trees $FT^{L-1,f}(u, v)$ with parameters $f, L-1$ as subproblems.

Assume we have already built the trees $FT^{i,f}(u, v)$, where $u, v \in V, 1 \leq i < L$, we describe how to build the trees $FT^{i+1,f}(u, v)$. Let (u, v, F) be a query for which we want to compute the distance $d^{i+1}(u, v, F)$ (as part of the construction of the tree $FT^{i+1,f}(u, v)$). Scan all the edges $(u, z) \in E$ and query the tree $FT^{i,f}(z, v)$ with the set F to find the distance $d^i(z, v, F)$. Querying the tree $FT^{i,f}(z, v)$ takes $O(f^2 \log i) = O(f^2 \log L)$ time as described in Lemma 17 (note that $f^2 \log L = \widetilde{O}(1)$ for $f \leq \log n$ as $L \leq n$), and we run $O(\text{out-degree(u)})$ such queries and take the minimum of the following equation.

$$d^{i+1}(u, v, F) = \min_z \{\omega(u, z) + d^i(z, v, F) \mid (u, z) \in E \;\; AND \;\; u, z, (u, z) \notin F\} \tag{1}$$

$$\text{parent}^{i+1}(u, v, F) = \arg\min_z \{\omega(u, z) + d^i(z, v, F) \mid (u, z) \in E \;\; AND \;\; u, z, (u, z) \notin F\} \tag{2}$$

Note that in Equation 1 we assume that for every vertex $u \in V$ it holds that G contains the self loops $(u, u) \in E$ such that $\omega(u, u) = 0$.

So the time to compute $d^{i+1}(u, v, F)$ is $\widetilde{O}(\text{out-degree(u)})$. Next, we describe how to reconstruct the path $P^{i+1}(u, v, F)$ in $O(L)$ additional time. We reconstruct the shortest path $P^{i+1}(u, v, F)$ by simply following the (at most L) parent pointers. In more details, let $z = \text{parent}^{i+1}(u, v, F)$ be the vertex defined according to Equation 2. We reconstruct the shortest path $P^{i+1}(u, v, F)$ by concatenating (u, z) with the shortest path $P^i(z, v, F)$ (which we reconstruct in the same way), thus we can reconstruct $P^{i+1}(u, v, F)$ edge by edge in constant time per edge, and hence it takes $O(L)$ time to reconstruct the path $P^{i+1}(u, v, F)$ that contains at most L edges.

The tree $FT^{i,f}(u, v)$ contains $i^f \leq L^f$ nodes, and thus all the trees $\{FT^{i,f}(u, v)\}$ for all $i \leq L, u, v \in V$ contain $O(n^2 L^{f+1})$ nodes together.

In each such node we compute the distance $d^i(u, v, \{a_1, \ldots, a_j\})$ in $\widetilde{O}(\text{out-degree(u)})$ time and reconstruct the path $P^i(u, v, \{a_1, \ldots, a_j\})$ in additional $O(L)$ time. Theretofore, computing all the distances $d^i(u, v, \{a_1, \ldots, a_j\})$ and all the paths $P^i(u, v, \{a_1, \ldots, a_j\})$ in all the nodes of all the trees $\{FT^{i,f}(u, v)\}_{u,v \in V, 1 \leq i \leq L}$ takes $\widetilde{O}(\sum_{i \leq L, u, v \in V} L^f(\text{out-degree(u)} + L)) = \widetilde{O}(mnL^{f+1} + n^2 L^{f+2})$ time. substituting $L = \widetilde{O}(n^{\epsilon/f})$ we get an algorithm to compute the trees $\{FT^{L,f}(u, v)\}_{u,v \in V}$ in $\widetilde{O}(mn^{1+\epsilon+\epsilon/f} + n^{2+\epsilon+2\epsilon/f})$ time.

This proves the following Lemma.

▶ **Lemma 18.** *One can deterministically construct the trees $FT^{L,f}(s, t)$ for every $s, t \in V$ in $\widetilde{O}(mn^{1+\epsilon+\epsilon/f} + n^{2+\epsilon+2\epsilon/f})$ time.*

In the full version we further reduce the runtime to $\widetilde{O}(mn^{1+\epsilon})$ by using dynamic programming only for computing the first $f-1$ levels of the trees $FT^{L,f}(s,t)$ and then applying Dijkstra in a sophisticated manner to compute the last layer of the trees $FT^{L,f}(s,t)$. In addition, we also boost-up the runtime of the greedy pivots selection algorithm from $\widetilde{O}(n^{2+\epsilon+\epsilon/f})$ to $\widetilde{O}(n^{2+\epsilon})$ time.

References

1. Udit Agarwal, Vijaya Ramachandran, Valerie King, and Matteo Pontecorvi. A Deterministic Distributed Algorithm for Exact Weighted All-Pairs Shortest Paths in Õ(n 3/2) Rounds. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 199–205, 2018.
2. N. Alon and J.H. Spencer. *The Probabilistic Method*. Fourth Edition. Wiley, 2016.
3. Noga Alon, Shiri Chechik, and Sarel Cohen. Deterministic combinatorial replacement paths and distance sensitivity oracles. *CoRR*, abs/1905.07483, 2019. `arXiv:1905.07483`.
4. Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Graph Expansion and Communication Costs of Fast Matrix Multiplication. *J. ACM*, 59(6):32:1–32:23, January 2013. `doi:10.1145/2395116.2395121`.
5. Austin R. Benson and Grey Ballard. A Framework for Practical Parallel Fast Matrix Multiplication. *SIGPLAN Not.*, 50(8):42–53, January 2015. `doi:10.1145/2858788.2688513`.
6. Aaron Bernstein. A Nearly Optimal Algorithm for Approximating Replacement Paths and K Shortest Simple Paths in General Graphs. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 742–755, 2010. URL: `http://dl.acm.org/citation.cfm?id=1873601.1873662`.
7. Aaron Bernstein and David Karger. Improved Distance Sensitivity Oracles via Random Sampling. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 34–43, 2008. URL: `http://dl.acm.org/citation.cfm?id=1347082.1347087`.
8. Aaron Bernstein and David Karger. A Nearly Optimal Oracle for Avoiding Failed Vertices and Edges. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing (STOC)*, pages 101–110, 2009. `doi:10.1145/1536414.1536431`.
9. Shiri Chechik, Sarel Cohen, Amos Fiat, and Haim Kaplan. $(1+\epsilon)$ approximate f-sensitive Distance Oracles. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 1479–1496, 2017. URL: `http://dl.acm.org/citation.cfm?id=3039686.3039782`.
10. Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. f-Sensitivity Distance Oracles and Routing Schemes. *Algorithmica*, 63(4):861–882, 2012. `doi:10.1007/s00453-011-9543-0`.
11. Camil Demetrescu and Mikkel Thorup. Oracles for Distances Avoiding a Link-failure. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 838–843, 2002. URL: `http://dl.acm.org/citation.cfm?id=545381.545490`.
12. 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, January 2008. `doi:10.1137/S0097539705429847`.
13. Ran Duan and Seth Pettie. Dual-failure Distance and Connectivity Oracles. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 506–515, 2009. URL: `http://dl.acm.org/citation.cfm?id=1496770.1496826`.
14. 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, September 2010. Appeared also in the Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '08). `doi:10.1145/1824777.1824784`.
15. David Eppstein. Finding the k Shortest Paths. *SIAM J. Comput.*, 28(2):652–673, 1998. `doi:10.1137/S0097539795290477`.

16 Zvi Gotthilf and Moshe Lewenstein. Improved Algorithms for the K Simple Shortest Paths and the Replacement Paths Problems. *Inf. Process. Lett.*, 109(7):352–355, March 2009. doi:10.1016/j.ipl.2008.12.015.

17 Fabrizio Grandoni and Virginia Vassilevska Williams. Improved Distance Sensitivity Oracles via Fast Single-Source Replacement Paths. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, 20-23, 2012*, pages 748–757, 2012. doi:10.1109/FOCS.2012.17.

18 J. Huang, L. Rice, D. A. Matthews, and R. A. v. d. Geijn. Generating Families of Practical Fast Matrix Multiplication Algorithms. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 656–667, May 2017. doi:10.1109/IPDPS.2017.56.

19 Neelesh Khanna and Surender Baswana. Approximate Shortest Paths Avoiding a Failed Vertex: Optimal Size Data Structures for Unweighted Graphs. In *27th International Symposium on Theoretical Aspects of Computer Science, STACS*, pages 513–524, 2010. doi:10.4230/LIPIcs.STACS.2010.2481.

20 Valerie King. Fully Dynamic Algorithms for Maintaining All-Pairs Shortest Paths and Transitive Closure in Digraphs. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 81–91, 1999. doi:10.1109/SFFCS.1999.814580.

21 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, April 2010. doi:10.1145/1721837.1721846.

22 Eugene L. Lawler. A Procedure for Computing the K Best Solutions to Discrete Optimization Problems and Its Application to the Shortest Path Problem. *Management Science*, 18(7):401–405, 1972. doi:10.1287/mnsc.18.7.401.

23 Cheng-Wei Lee and Hsueh-I Lu. Replacement Paths via Row Minima of Concise Matrices. *SIAM J. Discrete Math.*, 28(1):206–225, 2014. doi:10.1137/120897146.

24 K. Malik, A. K. Mittal, and S. K. Gupta. The K Most Vital Arcs in the Shortest Path Problem. *Oper. Res. Lett.*, 8(4):223–227, August 1989. doi:10.1016/0167-6377(89)90065-5.

25 Enrico Nardelli, Guido Proietti, and Peter Widmayer. A Faster Computation of the Most Vital Edge of a Shortest Path. *Inf. Process. Lett.*, 79(2):81–85, June 2001. doi:10.1016/S0020-0190(00)00175-7.

26 Liam Roditty and Uri Zwick. Replacement Paths and k Simple Shortest Paths in Unweighted Directed Graphs. In *Automata, Languages and Programming, 32nd International Colloquium, ICALP, 2005*, pages 249–260. See also ACM Trans. Algorithms, 8(4):33:1–11, 2012, 2005. doi:10.1007/11523468_21.

27 Oren Weimann and Raphael Yuster. Replacement Paths and Distance Sensitivity Oracles via Fast Matrix Multiplication. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 655–662, Washington, DC, USA, 2010. IEEE Computer Society. doi:10.1109/FOCS.2010.68.

28 Virginia Vassilevska Williams. Faster Replacement Paths. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, 23-25, 2011*, pages 1337–1346, 2011. doi:10.1137/1.9781611973082.102.

29 Virginia Vassilevska Williams and Ryan Williams. Subcubic Equivalences between Path, Matrix and Triangle Problems. In *51st Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010*, pages 645–654, 2010. doi:10.1109/FOCS.2010.67.

30 Jin Y. Yen. Finding the K Shortest Loopless Paths in a Network. *Management Science*, 17(11):712–716, 1971. doi:10.1287/mnsc.17.11.712.