2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS) | 978-1-6654-5519-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/FOCS54457.2022.00090

Algorithms and Lower Bounds for Replacement Paths under Multiple Edge Failures*

Virginia Vassilevska Williams
EECS
MIT
Cambridge, MA, USA
virgi@mit.edu

Eyob Woldeghebriel
EECS
MIT
Cambridge, MA, USA
eyobw@mit.edu

Yinzhan Xu
EECS
MIT
Cambridge, MA, USA
xyzhan@mit.edu

***Abstract*—This paper considers a natural fault-tolerant shortest paths problem: for some constant integer f, given a directed weighted graph with no negative cycles and two fixed vertices s and t, compute (either explicitly or implicitly) for every tuple of f edges, the distance from s to t if these edges fail. We call this problem f-Fault Replacement Paths (fFRP).**

We first present an $\tilde{O}(n^3)$ time algorithm for 2FRP in n-vertex directed graphs with arbitrary edge weights and no negative cycles. As 2FRP is a generalization of the well-studied Replacement Paths problem (RP) that asks for the distances between s and t for any single edge failure, 2FRP is at least as hard as RP. Since RP in graphs with arbitrary weights is equivalent in a fine-grained sense to All-Pairs Shortest Paths (APSP) [Vassilevska Williams and Williams FOCS'10, J. ACM'18], 2FRP is at least as hard as APSP, and thus a substantially subcubic time algorithm in the number of vertices for 2FRP would be a breakthrough. Therefore, our algorithm in $\tilde{O}(n^3)$ time is conditionally nearly optimal. Our algorithm immediately implies an $\tilde{O}(n^{f+1})$ time algorithm for the more general fFRP problem, giving the first improvement over the straightforward $O(n^{f+2})$ time algorithm.

Then we focus on the restriction of 2FRP to graphs with small integer weights bounded by M in absolute values. We show that similar to RP, 2FRP has a substantially subcubic time algorithm for small enough M. Using the current best algorithms for rectangular matrix multiplication, we obtain a randomized algorithm that runs in $\tilde{O}(M^{2/3}n^{2.9153})$ time. This immediately implies an improvement over our $\tilde{O}(n^{f+1})$ time arbitrary weight algorithm for all $f > 1$. We also present a data structure variant of the algorithm that can trade off pre-processing and query time. In addition to the algebraic algorithms, we also give an $n^{8/3-o(1)}$ conditional lower bound for combinatorial 2FRP algorithms in directed unweighted graphs, and more generally, combinatorial lower bounds for the data structure version of fFRP.

***Index Terms*—Replacement paths, fine-grained complexity.**

I. Introduction

Shortest paths problems are among the most basic problems in graph algorithms, and computer science in general. An important practically motivated version considers shortest paths computation in failure-prone graphs. The simplest such problem is the *Replacement Path* (RP) problem (studied e.g. by [1]–[13]) in which one is given a graph and two vertices s and t and one needs to return for every edge e, the shortest path from s to t in case edge e fails.

RP has several motivations. The first is, preparing for roadblocks or bad traffic in road networks, and similar situations in which edge links are no longer available. The second motivation is in Vickrey pricing for shortest paths in mechanism design (see [14], [15]).

It is natural to consider the generalization of RP to the case where up to f edges can fail for $f > 1$: given a graph G and two vertices s and t, for every set F of up to f failed edges, determine the distance between s and t in G with F removed. Let us call this the fFRP problem. Then similar to RP, fFRP is well-motivated: more than one roadblock can occur in road networks, and more than one link in a computer network can fail.

Since there are $\Theta(m^f)$ sets of f edges in an m edge graph, intuitively, the output of fFRP would have to be at least of size $\Theta(n^{2f})$ in a dense graph. However, just as in RP, one can show that the output only needs to be $\Theta(n^f)$.

Consider for instance $f = 2$. For every pair of edges (e, e'), at least one of them, say e should be on the shortest path P between s and t, as otherwise the s-t distance in $G \setminus F$ would be the same as that in G. Thus there are only $\leq n-1$ choices for e. Similarly, e' should be on the shortest path between s and t in $G \setminus \{e\}$, and there are only $\leq n-1$ choices for e'. Thus the output size of 2FRP is only $\Theta(n^2)$, and by a similar argument, the output size of fFRP is $\Theta(n^f)$.

It is natural to ask how close to n^f the best possible running time for fFRP can be. For graphs with nonnegative weights, fFRP can always be solved in $O(n^{f+2})$ time as follows. First, using Dijkstra's algorithm, compute in $O(n^2)$ time the shortest s-t path P. Then for every edge e on P, recursively solve $(f-1)$FRP in $G \setminus \{e\}$, where 0FRP is just a shortest paths computation.

Can we do better than $O(n^{f+2})$ time for fFRP in directed weighted graphs?

Let us consider RP (i.e. 1FRP). For graphs with m edges, n vertices, and arbitrary edge weights, RP can be solved in $\tilde{O}(m)$ time[1] in undirected graphs [2] and can be solved in

A full version of this paper can be found at https://arxiv.org/abs/2209.07016. Virginia Vassilevska Williams is supported by an NSF CAREER Award, NSF Grant CCF-2129139, a Google Research Fellowship and a Sloan Research Fellowship. Yinzhan Xu is supported by an NSF CAREER Award and NSF Grant CCF-2129139.

[1]The $\tilde{O}$ notation in this paper hides subpolynomial factors.

978-1-6654-5519-0/22/$31.00 ©2022 IEEE
DOI 10.1109/FOCS54457.2022.00090

$O(mn + n^2 \log\log n)$ time in directed graphs [3]. In dense directed graphs RP with arbitrary edge weights is subcubically equivalent to All-Pairs Shortest Paths (APSP) [5], [16], whose runtime has remained essentially cubic for 70 years, except for $n^{o(1)}$ factors.

Thus, (directed) fFRP for $f = 1$ does not have an $O(n^{f+2-\varepsilon})$ time algorithm for $\varepsilon > 0$, under the APSP hypothesis. Furthermore, there is no known $O(n^{f+2-\varepsilon})$ time algorithm for any $f > 1$.

The special case of $f = 2$, 2FRP, is particularly interesting since its output has size only $O(n^2)$. The APSP-based lower bound for RP implies that 2FRP also requires $n^{3-o(1)}$ time to solve. However, there is a gap between the best known upper bound of $O(n^4)$ and the $n^{3-o(1)}$ lower bound.

This brings us to the following open question, stated for instance in [17][2].

Can 2FRP be solved in essentially cubic time in directed weighted graphs?

Our first result is a resolution of the open question above for 2FRP.

Theorem 1. *In the Word-RAM Model with $O(\log n)$-bit words, the 2FRP problem on n-vertex $O(\log n)$-bit integer weighted directed graphs with no negative cycles can be solved in $O(n^3 \log^2 n)$ time by a deterministic algorithm or in $O(n^3 \log n)$ time by a randomized algorithm that succeeds with high probability.*

Since even RP is known to require $n^{3-o(1)}$ time under the APSP hypothesis [16], our algorithm for 2FRP is conditionally tight, up to $n^{o(1)}$ factors.

Moreover, our theorem has an immediate corollary for fFRP for $f > 2$ as well.

Corollary 2. *For all $f \geq 2$, fFRP in n-vertex directed weighted graphs with no negative cycles can be solved in $\tilde{O}(n^{f+1})$ time.*

Thus, for all $f > 1$, there is an algorithm for fFRP that runs polynomially faster than the trivial $O(n^{f+2})$ time, even though for $f = 1$ this was impossible under the APSP hypothesis!

For directed weighted graphs, replacement paths with vertex failures can be reduced to replacement paths with edge failures.[3] Thus, Corollary 2 works even if fFRP is replaced with f vertex-failure replacement paths.

For unweighted graphs, or graphs with small integer edge weights, there are better known running times for RP in directed graphs. Here there is a distinction between *algebraic* and *combinatorial* algorithms. While the terms themselves are not well-defined, combinatorial algorithms usually refer to algorithms that do not use algebraic techniques such as fast matrix multiplication while algebraic algorithms refer to algorithms that use such techniques. The study of combinatorial algorithms is motivated by the real-world inefficiency of fast matrix multiplication algorithms and by a desire to get algorithms that can perform better on sparser graphs, as it is usually difficult for algebraic algorithms to take full advantage of the sparsity of graphs.

In the case of unweighted graphs, directed RP has a combinatorial algorithm with an $\tilde{O}(m\sqrt{n})$ running time [10], which is essentially optimal as any further improvement would imply a breakthrough in Boolean Matrix Multiplication [5], [16]. For graphs with small integer edge weights in the range $\{-M, \ldots, M\}$, there is an algebraic algorithm for directed RP with an $\tilde{O}(Mn^\omega)$ running time [12], [13], where $\omega \in [2, 2.373)$ denotes the exponent for multiplying two $n \times n$ matrices [18]–[20].

Can we solve fFRP in subcubic time in graphs with bounded integer weights when $f > 1$? Due to the output size, fFRP for $f \geq 3$ cannot have a subcubic time algorithm. Thus the only generalization of RP that can still have a subcubic time algorithm in bounded weight graphs is 2FRP.

Does 2FRP in graphs with small integer weights have a truly subcubic time algorithm?

Similar to Corollary 2, if 2FRP in small weight graphs has an $O(n^{3-\varepsilon})$ time algorithm, then for every $f \geq 2$, we would get an $O(n^{f+1-\varepsilon})$ time algorithm for fFRP in small weight graphs, which is close to the output size $\Theta(n^f)$ and always better than our new $\tilde{O}(n^{f+1})$ time algorithm for the arbitrary weight case.

A reason to believe that a subcubic time algorithm may be possible for 2FRP is that the output size is only quadratic. Another generalization of RP that has only quadratic output size, the Single Source Replacement Paths problem (SSRP), can be solved in $\tilde{O}(Mn^\omega)$ time for graphs with edge weights in $\{1, \ldots, M\}$ [13] or in $O(M^{0.8043}n^{2.4957})$ time for graphs with edge weights in $\{-M, \ldots, M\}$ [21]. Hence it is possible that 2FRP also has a subcubic time algorithm for small weight graphs.

Meanwhile, current techniques do not seem to yield subcubic time algorithms: replacement paths problems have been studied (e.g. in [22]) as a special case of f-failure distance sensitivity oracles (DSO), which are data structures that can support replacement path queries for any set of f edge faults. To solve 2FRP, one would need to pre-process a 2-failure DSO and then perform $O(n^2)$ queries on it, assuming that the $O(n^2)$ queries are known.

The best known DSO for graphs with small integer edge weights and more than one fault is by van den Brand and Saranurak [23]. For every $\alpha \in [0, 1]$, their oracle can achieve $\tilde{O}(Mn^{\omega+(3-\omega)\alpha})$ pre-processing time and $\tilde{O}(Mn^{2-\alpha})$ query time for any constant number of failures on n-vertex graphs with edge weights in $\{-M, \ldots, M\}$. Balancing the pre-

[2]Bhosle and Gonzalez [17] claimed an $O(n^3)$ time algorithm for the special case where both failed edges are on the original shortest path. However, their approach doesn't quite work as written, and we include a discussion about the issue in the full version.

[3]Given a graph $G = (V, E)$, we create a graph G' as follows. For every $v \in V$, we create two vertices v_{in} and v_{out} in G', and add an edge $(v_{\text{in}}, v_{\text{out}})$ with weight 0 to G'. For every $(u, v) \in E$ with weight w, we add an edge $(u_{\text{out}}, v_{\text{in}})$ to G' with weight w. Let $U \subseteq V$ be any set of f failed vertices in G. We define e_U to be $\{(u_{\text{in}}, u_{\text{out}}) : u \in U\}$. For any $s, t \in V \setminus U$, it is not difficult to verify that $d_{G\setminus U}(s, t) = d_{G'\setminus e_U}(s_{\text{in}}, t_{\text{out}})$, which completes the reduction.

processing time and the $O(n^2)$ queries needed results in an $\tilde{O}(Mn^3)$ running time for 2FRP – a running time that is *never* subcubic, even if $M = O(1)$.

We overcome the difficulties that come from using existing DSO techniques, and are able to provide a new sensitivity oracle for 2FRP with fast pre-processing and query times.

Theorem 3. *For any given positive integer parameter $g \leq O(n)$, there exists a data structure that can pre-process a given directed graph G with integer edge weights in $\{-M, \ldots, M\}$ and no negative cycles and fixed vertices s and t, in $\tilde{O}(Mn^{\omega+1}/g + Mn^{2.8729})$ time, and can answer queries of the form $d_{G\setminus\{e_1,e_2\}}(s,t)$ in $\tilde{O}(g^2)$ time. This data structure has randomized pre-processing which succeeds with high probability. The size of the data structure is $\tilde{O}(n^{2.5})$.*

If the edge weights of G are positive, the pre-processing time and the size of the data structure can be improved to $\tilde{O}(Mn^{\omega+1}/g + M^{0.3544}n^{2.7778} + Mn^{2.5794})$ and $\tilde{O}(n^2)$ respectively.

We note that the running time exponents shown above with 4 digits after the decimal points all follow from fast rectangular matrix multiplication [24]. As a corollary we obtain the first truly subcubic time algorithm for 2FRP in graphs with bounded integer weights.

Corollary 4. *The 2FRP problem on n-vertex directed graphs with integer edge weights in $\{-M, \ldots, M\}$ and with no negative cycles can be solved in $\tilde{O}(M^{2/3}n^{2.9153})$ time by a randomized algorithm that succeeds with high probability.*

Our new algorithms for 2FRP for bounded and arbitrary weight graphs are interesting as they show that 2FRP is not much more difficult than RP– both admit a cubic time algorithm for general graphs and subcubic time algorithms for bounded integer weight graphs.

We also immediately obtain the following corollary for $f > 2$, beating our algorithm for the arbitrary weight case for small enough M:

Corollary 5. *For any $f \geq 2$, fFRP on n-vertex directed graphs with integer edge weights in $\{-M, \ldots, M\}$ and with no negative cycles can be solved in $\tilde{O}(M^{2/3}n^{f+0.9153})$ time by a randomized algorithm that succeeds with high probability.*

We remark that, if given the failed edges, all (except one) of our algorithms are able to report an optimal replacement path P in $\tilde{O}(|P|)$ time. The exception is the positive weight case in Theorem 3, as it uses Gu and Ren's DSO [25] as a subroutine, which does not support path reporting.

So far our algorithms have used fast matrix multiplication. Often one desires more practical combinatorial algorithms. How fast can combinatorial algorithms for fFRP in unweighted graphs be?

Since the output size for fFRP with $f > 2$ is supercubic, only distance sensitivity oracles can give subcubic bounds. We show (conditionally) that any combinatorial f-failure sensitivity oracle for a fixed pair of vertices that can answer queries faster than running Dijkstra's algorithm at each query, must have high pre-processing time.

Our conditional lower bound is based on the Boolean Matrix Multiplication (BMM) hypothesis which says that any "combinatorial" algorithm for $n \times n$ Boolean matrix multiplication requires $n^{3-o(1)}$ time. By [5], [16], the BMM hypothesis is equivalent to the hypothesis which states that any combinatorial algorithm for Triangle Detection, which asks whether a given graph contains a triangle, in n-vertex graphs requires $n^{3-o(1)}$ time.

Theorem 6. *Let $k \geq 1$ be any constant integer. Suppose that there is a combinatorial data structure that can pre-process any directed unweighted n-vertex graph G and fixed vertices s, t in $\tilde{O}(n^{2+k/(k+1)-\epsilon})$ time, and can then answer k-fault distance sensitivity queries between s and t in $\tilde{O}(n^{2-\epsilon})$ time, for $\epsilon > 0$. Then there is a combinatorial algorithm for Triangle Detection running in $\tilde{O}(n^{3-\epsilon})$ time.*

The proof of the theorem can be found in the full version. For $k = 2$, the above theorem implies that (combinatorial) 2FRP requires $n^{8/3-o(1)}$ time under the BMM hypothesis. This means that RP is slightly easier than 2FRP in the combinatorial setting since it has an $\tilde{O}(n^{2.5})$ time algorithm [10].

We leave it as an open problem to obtain an $\tilde{O}(n^{8/3})$ time combinatorial algorithm for unweighted 2FRP. One reason to suspect the existence of such an algorithm is the existence of 2-fault-tolerant BFS trees of size $O(n^{5/3})$ [26]. Note that for RP there are fault-tolerant BFS trees of size $O(n^{3/2})$ and a combinatorial algorithm of runtime $\tilde{O}(n^{5/2})$ [10], [27], though there is no direct reduction between algorithm running times and the sparsity of fault-tolerant subgraphs.

Related work: The running time for APSP in a graph with arbitrary polylog(n) bit integers has remained essentially cubic in the number of vertices for almost 70 years, with the current best running time being $n^3/2^{\Omega(\sqrt{\log n})}$ by Williams [28], [29]. A truly subcubic time algorithm for APSP with arbitrary weights would be a significant breakthrough.

The restriction of APSP to graphs with small integer edge weights does have truly subcubic algorithms. Seidel gave an $\tilde{O}(n^\omega)$ time algorithm for APSP in an undirected unweighted graph [30]. This algorithm was later generalized by Shoshan and Zwick to yield an $\tilde{O}(Mn^\omega)$ time algorithm for APSP in an undirected graph with integer edge weights in $\{0, 1, \ldots, M\}$ [31]. For directed graphs with edge weights in $\{-M, \ldots, M\}$, the current best algorithm is by Zwick [32] that runs in $\tilde{O}(M^{1/(4-\omega)}n^{2+1/(4-\omega)})$ time, or $\tilde{O}(M^{0.7519}n^{2.5286})$ time using the current best algorithm for rectangular matrix multiplication [19].

For graphs with m edges, n vertices and arbitrary integer weights, RP can be solved in $\tilde{O}(m)$ time in undirected graphs [2], and in $O(mn + n^2 \log\log n)$ time in directed graphs [3]. For graphs with small integer weights in $\{-M, \ldots, M\}$, Vassilevska Williams [12] showed an $\tilde{O}(Mn^\omega)$ time randomized algorithm for RP. For unweighted directed graphs, there is an $\tilde{O}(m\sqrt{n})$ time deterministic combinatorial algorithm for RP [33]. RP has also been studied in

the approximate setting [4], in planar graphs [6], [7] and in DAGs [8], [9].

In the more general Single-Source Replacement Paths (SSRP) problem, we are asked to compute all the replacement path distances for a single source s but for all possible targets t and all possible edges e on one s to t shortest path. Grandoni and Vassilevska Williams [13], [34] generalized the RP algorithm of Vassilevska Williams [12] to compute SSRP. Their randomized algorithm runs in $\tilde{O}(Mn^{\omega})$ time for graphs with weights in $\{1,\ldots,M\}$ and in $\tilde{O}(M^{0.7519}n^{2.5286})$ time for graphs with weights in $\{-M,\ldots,M\}$. The latter case was recently improved by Gu, Polak, Vassilevska Williams and Xu to $O(M^{0.8043}n^{2.4957})$ time [21]. For unweighted directed graphs with n vertices and m edges, Chechik and Magen [35] showed an $\tilde{O}(m\sqrt{n}+n^2)$ time combinatorial algorithm and an $mn^{1/2-o(1)}$ conditional lower bound for combinatorial algorithms.

There is a significant body of work on single-fault distance sensitivity oracle. The first nontrivial DSO for weighted graphs was given by Demetrescu, Thorup, Chowdhury and Ramachandran [36], who gave a deterministic oracle with constant query time and $O(n^{3.5})$ construction time. They also had an alternative DSO that needs $O(n^4)$ construction time, but only uses $\tilde{O}(n^2)$ space and keeps constant query time. Bernstein and Karger [37] gave a deterministic DSO for weighted graphs with $\tilde{O}(n^3)$ pre-processing time and $\tilde{O}(1)$ query time. This is essentially optimal barring improvements in APSP. On the other hand, single-fault DSOs for graphs with small integer edge weights do not have a (conditionally) optimal algorithm. The first DSO for small integer weighted graphs with subcubic pre-processing time and sublinear query time is given by Grandoni and Vassilevska Williams [13], [34]. Their DSO for directed graphs with integer edge weights in $\{-M,\ldots,M\}$ has an $\tilde{O}(Mn^{\omega+1/2}+Mn^{\omega+\alpha(4-\omega)})$ pre-processing time and an $\tilde{O}(n^{1-\alpha})$ query time for any parameter $\alpha\in[0,1]$. Chechik and Cohen improved the DSO to $\tilde{O}(Mn^{2.873})$ pre-processing time and $\tilde{O}(1)$ query time [38]. An algorithm by Ren improves the pre-processing time to $\tilde{O}(Mn^{2.733})$ and the query time to $O(1)$, but it only works for graphs with positive integer weights in $\{1,\ldots,M\}$ [39]. Gu and Ren [25] recently improved the pre-processing time to $\tilde{O}(Mn^{2.5794})$ for constructing DSO for such graphs.

The first major step in multiple-fault DSOs was a DSO by Weimann and Yuster [22] which can efficiently handle up to $f=O(\log n/\log\log n)$ edge failures (for larger number of failures it will not be faster than brute-force). Their DSO is randomized, and has an $\tilde{O}(Mn^{\omega+1-\alpha})$ pre-processing time and an $\tilde{O}(n^{2-(1-\alpha)/f})$ query time for graphs with weights in $\{-M,\ldots,M\}$, for any chosen parameter $\alpha\in(0,1)$. They also have an alternative DSO which has $\tilde{O}(M^{0.68}n^{3.529-\alpha})$ pre-processing time and $\tilde{O}(n^{2-2(1-\alpha)/f})$ query time for any chosen parameter $\alpha\in(0,1)$ using the current best algorithm for rectangular matrix multiplication [24]. For graphs with arbitrary edge weights, their DSO has an $\tilde{O}(n^{4-\alpha})$ pre-processing time and an $\tilde{O}(n^{2-2(1-\alpha)/f})$ query time [22]. Their DSO for graphs with arbitrary edge weights was later derandomized by Alon, Chechik, and Cohen in [33]. The current best multiple-fault DSO for small integer weighted graphs is a randomized DSO by van den Brand and Saranurak, which has an $\tilde{O}(Mn^{\omega+(3-\omega)\alpha})$ pre-processing time and an $\tilde{O}(Mn^{2-\alpha}f^2+Mnf^{\omega})$ query time, for any parameter $\alpha\in[0,1]$ [23].

Duan and Pettie designed an $\tilde{O}(n^2)$ space two-fault DSO with $\tilde{O}(1)$ query time [40]. Since their focus is space complexity instead of pre-processing time, their result is not directly comparable to ours. Recently, Duan and Ren [41] generalized [40] to f failures: they designed an $\tilde{O}(fn^4)$ space f-fault DSO with $f^{O(f)}$ query time, though it only works for undirected weighted graphs.

II. Preliminaries

Throughout this paper, use $\pi_G(u,v)$ to denote a shortest path from u to v in G and use $d_G(u,v)$ to denote its distance. We also use $\pi_G(u,v,e)$ to denote a shortest path from u to v in the graph G with edge e removed, and use $d_G(u,v,e)$ to denote its distance. We sometimes drop the subscript G if it is clear from the context. All graphs considered in this paper don't have negative cycles.

Let s,t be the source and target vertices in a graph G and let $\pi_G(s,t)$ be a shortest path from s to t in G. Suppose we remove a set of edges S from G. We say a path P is *canonical* (with respect to $\pi_G(s,t)$ and S) if for any vertices u,v that appear both on P and on $\pi_G(s,t)$ such that u appears before v in both P and $\pi_G(s,t)$ and the subpath from u to v on $\pi_G(s,t)$ is not disconnected by S, then the subpath from u to v in P is the same as the subpath from u to v in $\pi_G(s,t)$. Clearly, at least one of the replacement path $\pi_{G\setminus S}(s,t)$ is canonical. This is a light-weighted tie-breaking scheme. See Figure 1a and Figure 1b for examples.

For a graph G, we use $\widehat{G}$ to denote a copy of G with all directions of the edges reversed. We use $\widehat{G}$ for notational succinctness. For instance, instead of saying computing single-target replacement paths to t in G, we can say computing SSRP from t in $\widehat{G}$.

We use $\omega<2.3729$ [18]–[20] to denote the square matrix multiplication exponent. For any $k>0$, we also use $\omega(1,k,1)$ to denote the exponent for multiplying an $n\times n^k$ matrix and an $n^k\times n$ matrix. Currently, the fastest algorithm for rectangular matrix multiplication is by Le Gall and Urrutia [24]. It is well-known that the function $\omega(1,k,1)$ with respect to k is convex when $k>0$ (see e.g. [42]).

III. Technical Overview

In this section, we describe the high-level ideas and key components in our algorithms for 2FRP in arbitrary weighted graphs and in small integer weighted graphs.

Let $G=(V,E)$ be a weighted graph with no negative cycles, and let $s,t\in V$ be the fixed source and target of the 2FRP instance. Let $\pi_G(s,t)$ be a shortest path from s to t. Both algorithms handle the following cases of two-fault replacement paths queries separately: the case where only one

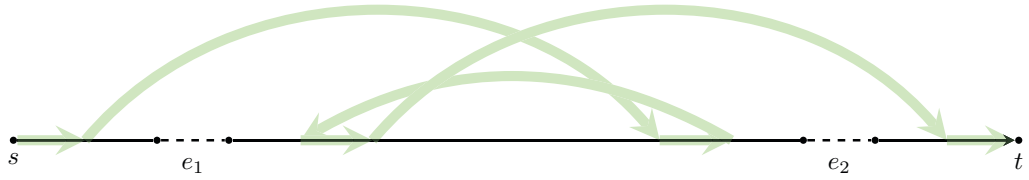

Fig. 1a. An example of a canonical path. The horizontal line (including e_1 and e_2) is the shortest path $\pi_G(s,t)$ from s to t in G, and the set of edges we remove is $S = \{e_1, e_2\}$.

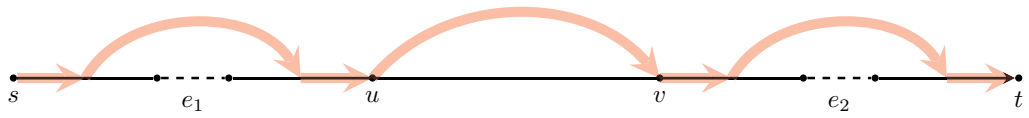

Fig. 1b. An example of a non-canonical path. The horizontal line (including e_1 and e_2) is the shortest path $\pi_G(s,t)$ from s to t in G, and the set of edges we remove is $S = \{e_1, e_2\}$. The path is not canonical because the subpath from u to v on $\pi_G(s,t)$ is not disconnected by $\{e_1, e_2\}$ while the path is not using that subpath.

of the two failed edges e_1, e_2 is on the original shortest path $\pi_G(s,t)$, and the case where both failed edges are on $\pi_G(s,t)$.

First, we consider the case where only one of the two failed edges e_1, e_2, say e_1, is on $\pi_G(s,t)$. Let $H = G \setminus \{e_2\}$. We aim to compute $d_{G\setminus\{e_1,e_2\}}(s,t) = d_{H\setminus\{e_1\}}(s,t)$, i.e., a one-fault replacement path query in H. Since e_2 is not on $\pi_G(s,t)$, $\pi_G(s,t)$ is still a shortest path from s to t in H. Given a shortest path $\pi_H(s,t) = \pi_G(s,t)$, the structure of one-fault replacement paths is well-understood. It is known (see e.g. [22]) that one of the optimal one-fault replacement paths shares a prefix and a suffix with the shortest path, and contains a detour part that connects the prefix and the suffix. Importantly, the detour part does not use any edge on the original shortest path $\pi_H(s,t) = \pi_G(s,t)$. Therefore, in order to understand the distances of the detours, we need to compute the distance in the graph $H \setminus \pi_G(s,t)$,[4] which is exactly the graph $G \setminus \pi_G(s,t) \setminus \{e_2\}$. Thus, the detour distances can be efficiently computed by a one-fault DSO on the graph $G \setminus \pi_G(s,t)$. Based on this intuition, a key component in both of our algorithms is a one-fault DSO on the graph $G\setminus\pi_G(s,t)$. Depending on the range of edge weights of the input graph, we will use different DSOs accordingly.

One-fault DSO does not seem to help the case where both failed edges e_1, e_2 are on $\pi_G(s,t)$. The structure of $d_{G\setminus\{e_1,e_2\}}(s,t)$ is more complicated than the structure of one-fault replacement paths. One can show that one optimal 2-fault replacement path still shares a prefix and a suffix with $\pi_G(s,t)$, but the middle part between the prefix and the suffix is not simply a detour that does not use any edge on $\pi_G(s,t)$. In fact, it is possible that the middle part enters and leaves the subpath of $\pi_G(s,t)$ between e_1 and e_2 an arbitrary number of times, as shown in Figure 2b. To understand the middle part better, we study the following problem as a key subroutine in our algorithms: for every two vertices u, v on $\pi_G(s,t)$ where u appears earlier than v, we aim to compute $f(v,u)$ which is defined as $d_{G\setminus\pi_G(s,u)\setminus\pi_G(v,t)}(v,u)$, i.e., the distance of a shortest path from v to u that is not allowed to use edges before u or after v on $\pi_G(s,t)$. Intuitively, $f(v,u)$ captures the structure of the middle part of $d_{G\setminus\{e_1,e_2\}}(s,t)$, as the optimal path for $f(v,u)$ can also enter and leave the shortest path $\pi_G(s,t)$ multiple times. In the full version, we will give efficient algorithms for computing these distances $f(v,u)$ in both arbitrary weighted graphs and small integer weighted graphs. The running times of these two algorithms are summarized below.

Lemma 7. *There exists a deterministic algorithm that can compute f in n-vertex weighted graphs with no negative cycles in $O(n^3)$ time.*

Lemma 8. *There exists a randomized algorithm that can compute f in n-vertex graphs with integer edge weights in $\{-M, \ldots, M\}$ in $\tilde{O}(M^{1/3}n^{2+\omega/3})$ time. Using rectangular matrix multiplication, the running time improves to $\tilde{O}(M^{0.3544}n^{2.7778})$.*

Our algorithm for 2FRP on small integer weighted graphs needs to run SSRP multiple times on different subgraphs of G (and related graphs) with different sources (see Section V-A for an overview of the algorithm). However, the best running time for SSRP on graphs with integer edge weights in $\{-M, \ldots, M\}$ is $O(M^{0.8043}n^{2.4957})$ by Gu, Polak, Vassilevska Williams and Xu [21]. Simply running their algorithm the required amount of times easily exceeds the running time we aim for. Fortunately, in most of our SSRP computations, we only need the replacement path distances $d_{G'\setminus\{e\}}(s,t)$ for $t \in T$ and $e \in \pi_{G'}(s,t)$, where the size of T is much smaller than n. In the full version, we will adapt Grandoni

[4]Throughout this paper, a path P formally denotes the set of its edges. Thus, $H \setminus \pi_G(s,t)$ is a subgraph of H that removes all edges on the s to t shortest path, but keeps all the vertices on it.

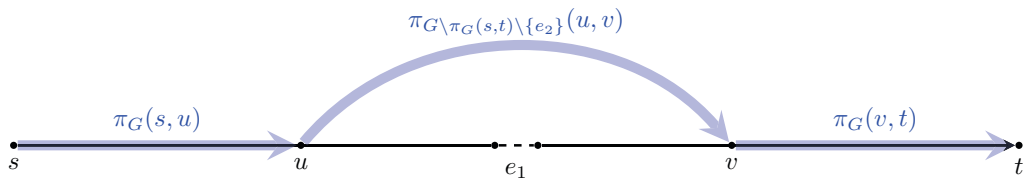

Fig. 2a. A typical 2-fault replacement path where e_1 is on the original shortest path while e_2 is not.

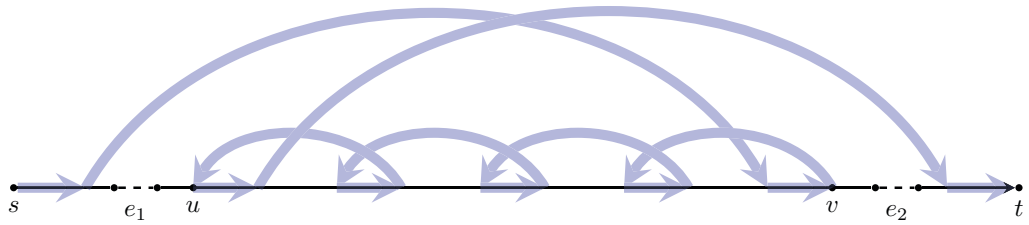

Fig. 2b. A typical 2-fault replacement path where both e_1 and e_2 are on the original shortest path. All paths shown that do not lie on the original shortest path do not use any edge on it.

and Vassilevska Williams's algorithm for SSRP on graphs with integer edge weights in $\{-M,\ldots,M\}$ [13] to achieve a more efficient algorithm when T is small:

Lemma 9. *Given an n-vertex graph G whose edge weights are in $\{-M,\ldots,M\}$, a source vertex $s \in V(G)$, and a subset $T \subseteq V(G)$, there is a randomized algorithm that computes $d_G(s,t,e)$ for every $t \in T$ and $e \in \mathcal{T}_s$ where $\mathcal{T}_s$ is a shortest path tree rooted at s in $\tilde{O}(Mn^{\omega} + M^{\frac{1}{4-\omega}} n^{1+\frac{1}{4-\omega}} \cdot |T|)$ time with high probability.*

Note that we can potentially use ideas from [21] to make Lemma 9 faster for large enough T, but this lemma won't be a bottleneck of our whole algorithm. In fact, for the value of $|T|$ we end up using for our 2FRP algorithm, the Mn^{ω} term in the above lemma dominates the other term, and [21]'s techniques cannot avoid the Mn^{ω} term either. Therefore, we choose to adapt the simpler algorithm by Grandoni and Vassilevska Williams [13].

All (except the positive weight case of Theorem 3) of our algorithms can be used to report paths efficiently, by using known techniques for finding solutions of dynamic programming and finding witnesses for matrix multiplication problems [43].

IV. Nearly Cubic Time Algorithm for Weighted Graphs

In this section, we show our $\tilde{O}(n^3)$ time algorithm for 2FRP. We use two drastically different algorithms for the case where only one failed edge is on the original s to t shortest path and the case where both failed edges are on the original shortest path.

When only one failed edge is on the original s to t shortest path, our algorithm is essentially a simple reduction to the (one-fault) distance sensitivity oracle problem. For the other case where both failed edges are on the shortest path, we carefully design algorithms that can capture the patterns of optimal replacement paths.

A. Only One Failed Edge on Original Shortest Path

Let $G = (V, E)$ be our input graph, and let $\pi_G(s,t)$ be a shortest path from s to t in G. We will compute all replacement path distances $d_{G\setminus\{e_1,e_2\}}(s,t)$ for $e_1 \in \pi_G(s,t)$ and $e_2 \notin \pi_G(s,t)$. Our algorithm relies on the following efficient data structure for one-failure distance sensitivity oracle by Bernstein and Karger [37].

Theorem 10 ([37]). *Given a weighted graph $H = (V, E)$ with n vertices and no negative cycles, there exists a deterministic data structure that can pre-process H in $O(n^3 \log^2 n)$ time and then answer queries in the form $d_{H\setminus\{e\}}(u,v)$ for any $u, v \in V$ and $e \in E$ in $O(1)$ time. Allowing randomized data structure that succeeds with high probability, the pre-processing time can be improved to $O(n^3 \log n)$.*

Even though they only stated their DSO for graphs with nonnegative edge weights, their DSO also works for graphs with possibly negative edge weights but no negative cycles, after an $O(n^3)$ pre-processing step that replaces all edges with nonnegative edges [44], [45].

Given our graph G, we create another graph G' with $O(n)$ vertices. First, we copy G to G' and remove all edges on $\pi_G(s,t)$. Let $h = O(n)$ be the number of vertices on $\pi_G(s,t)$ and let $p_1,\ldots,p_h$ be vertices on the path $\pi_G(s,t)$, in order they appear on $\pi_G(s,t)$. In particular, $s = p_1$ and $t = p_h$. We add h vertices $a_1,\ldots,a_h$ to G'. For any $1 \le i \le j \le h$, we add an edge from a_j to p_i with weight $d_G(s,p_i)$. Finally, we add another h vertices $b_1,\ldots,b_h$ and for any $1 \le i \le j \le h$, we add an edge from p_j to b_i with weight $d_G(p_j,t)$.

The following lemma relates $d_{G\setminus\{e_1,e_2\}}(s,t)$ with replacement path distances in G'.

Lemma 11. *For any $e_1 = (p_i, p_{i+1}) \in \pi_G(s,t)$ and any $e_2 \notin \pi_G(s,t)$,*

$$d_{G\setminus\{e_1,e_2\}}(s,t) = d_{G'\setminus\{e_2\}}(a_i, b_{i+1}).$$

Proof: First, we notice that $\pi_G(s,t)$ is still a shortest path from s to t in $G \setminus \{e_2\}$. Also, $\pi_{G\setminus\{e_1,e_2\}}(s,t)$ is a one-fault replacement path on the graph $G \setminus \{e_2\}$. It is well-known that (at least one) one-fault replacement path consists of the following three parts: a prefix that is a prefix of the original shortest path, a detour that does not use any edge on the original shortest path, and a suffix that is also a suffix of the original shortest path (see e.g. [13]). Therefore, $d_{G\setminus\{e_1,e_2\}}(s,t)$, when viewed as a one-fault replacement path in $G \setminus \{e_2\}$ can be expressed as

$$\begin{aligned} d_{G\setminus\{e_1,e_2\}}(s,t) = &\min_{1\le x\le i<y\le h} \big\{d_{G\setminus\{e_2\}}(s,p_x) \\ &+d_{G\setminus\{e_2\}\setminus\pi_G(s,t)}(p_x,p_y) + d_{G\setminus\{e_2\}}(p_y,t)\big\} \\ = &\min_{1\le x\le i<y\le h} \big\{d_G(s,p_x) \\ &+d_{G\setminus\{e_2\}\setminus\pi_G(s,t)}(p_x,p_y) + d_G(p_y,t)\big\}. \end{aligned}$$

On the other hand, we consider $d_{G'\setminus\{e_2\}}(a_i, b_{i+1})$. Clearly, a_i must first go to some neighbor p_x for some $x \le i$ where the edge weight of (a_i, p_x) is $d_G(s,p_x)$. Similarly, the last edge on any a_i to b_{i+1} path must travel from a neighbor of b_{i+1} to b_{i+1}. Thus, the last edge must be from p_y for some $y \ge i+1$ with weight $d_G(p_y,t)$. Also, the subpath from p_x to p_y lies entirely in $G' \setminus \{e_2\}$; this subpath cannot use any vertex a_j or b_j for $1 \le j \le h$ either because these vertices either have 0 out-degree or 0 in-degree. Thus, the subpath from p_x to p_y actually lies entirely in $G \setminus \{e_2\} \setminus \pi_G(s,t)$. We can therefore express $d_{G'\setminus\{e_2\}}(a_i, b_{i+1})$ as

$$\begin{aligned} d_{G'\setminus\{e_2\}}(a_i, b_{i+1}) = &\min_{1\le x\le i<y\le h} \{d_G(s,p_x) \\ &+d_{G\setminus\{e_2\}\setminus\pi_G(s,t)}(p_x,p_y) + d_G(p_y,t)\big\}, \end{aligned}$$

which matches exactly with the formula for $d_{G\setminus\{e_1,e_2\}}(s,t)$. ∎

Using Lemma 11 and Theorem 10, we can easily solve the case where only one failed edge is on $\pi_G(s,t)$ in $\tilde{O}(n^3)$ time. We first construct G' and use Theorem 10 to pre-process G'. Then for any two-fault replacement path query $d_{G\setminus\{e_1,e_2\}}(s,t)$ where $e_1 \in \pi_G(s,t)$ and $e_2 \notin \pi_G(s,t)$, we query $d_{G'\setminus\{e_2\}}(a_i, b_{i+1})$ from the DSO in $O(1)$ time. By Lemma 11, this distance equals $d_{G\setminus\{e_1,e_2\}}(s,t)$. Since there are only $O(n^2)$ queries, the pre-processing is the bottleneck and thus this case takes $O(n^3 \log^2 n)$ deterministic time or $O(n^3 \log n)$ randomized time with high probability.

B. Both Failed Edges on Original Shortest Path

In this section, we will describe an algorithm that computes all replacement path distances $d_{G\setminus\{e_1,e_2\}}(s,t)$ for $e_1, e_2 \in \pi_G(s,t)$. Again, we let $p_1, \ldots, p_h$ be vertices on the path $\pi_G(s,t)$, in the order they appear on $\pi_G(s,t)$. Without loss of generality, $e_1 = (p_i, p_{i+1})$ and $e_2 = (p_j, p_{j+1})$ for some $1 \le i < j < h$.

Let P be a shortest path from s to t in $G \setminus \{e_1, e_2\}$ that is canonical (recall the definition of canonical in Section II). There are essentially two cases in this section: the case where P does not use any vertex on $\pi_G(s,t)$ between e_1 and e_2 and the case where P uses at least one such vertex.

We first consider the case where P does not use any vertex on $\pi_G(s,t)$ between e_1 and e_2. This can be thought as a generalization of the RP algorithm in [3].

Lemma 12. *In $O(n^3)$ time, we can compute the replacement path distances $d_{G\setminus\{e_1,e_2\}}(s,t)$ for every pair of $e_1, e_2 \in \pi_G(s,t)$ where $e_1 = (p_i, p_{i+1})$ and $e_2 = (p_j, p_{j+1})$ for some $1 \le i < j < h$ and the replacement path does not use any vertex p_k for $i < k \le j$.*

Proof: First we fix some e_1, e_2 and consider their corresponding replacement path P. Without loss of generality, we assume P is canonical. Let p_x be the rightmost vertex (furthest from s) P uses on $\pi_G(s,t)$ before e_1. Similarly, let p_y be the leftmost vertex (furthest from t) P uses on $\pi_G(s,t)$ after e_2.

Since P is canonical, its subpath from s to p_x must use the portion from s to p_x on $\pi_G(s,t)$ and thus has length $d_G(s,p_x)$. Also, it implies that p_y must appear after p_x on P. Similarly, the subpath of P from p_y to t must use the portion from p_y to t on $\pi_G(s,t)$ and thus has length $d_G(p_y,t)$.

Now we consider the subpath of P from p_x to p_y. It cannot use any edge between s and p_x on $\pi_G(s,t)$, since otherwise, the subpath from s to this edge does not match the portion from s to this edge on $\pi_G(s,t)$, making P not canonical. Similarly, it cannot use any edge between p_y and t on $\pi_G(s,t)$. The subpath of P from p_x to p_y cannot use any edge between p_x and e_1 on $\pi_G(s,t)$ either, due to the definition of p_x. Similarly, it cannot use any edge between e_2 and p_y. We also assumed that P does not use any vertex between e_1 and e_2 on $\pi_G(s,t)$, and thus it does not use any edge between e_1 and e_2 either. Therefore, the subpath from p_x to p_y completely avoids $\pi_G(s,t)$ and thus its length is $d_{G\setminus\pi_G(s,t)}(p_x,p_y)$.

Thus, we have shown that $d_{G\setminus\{e_1,e_1\}}(s,t) = d_G(s,p_x) + d_{G\setminus\pi_G(s,t)}(p_x,p_y) + d_G(p_y,t)$. In general, for any $e_1 = (p_i, p_{i+1})$ and $e_2 = (p_j, p_{j+1})$ for some $1 \le i < j < h$,

$$\begin{aligned} d_{G\setminus\{e_1,e_2\}}(s,t) = &\min_{\substack{1\le x\le i \\ j+1\le y\le h}} \{d_G(s,p_x) \\ &+d_{G\setminus\pi_G(s,t)}(p_x,p_y) + d_G(p_y,t)\big\}, \end{aligned}$$

as long as the replacement path does not use any vertex p_k, for $i < k \le j$.

Let $T(x,y)$ be $d_G(s,p_x) + d_{G\setminus\pi_G(s,t)}(p_x,p_y) + d_G(p_y,t)$. After running APSP in $G \setminus \pi_G(s,t)$ in $O(n^3)$ time, we can compute all values $T(x,y)$ and store points (x,y) in a 2D range tree and associate a value $T(x,y)$ with point (x,y) in $\tilde{O}(n^2)$ time, so that the 2D range tree can support orthogonal range minimum queries. Then for any $e_1 = (p_i, p_{i+1})$ and $e_2 = (p_j, p_{j+1})$ for some $1 \le i < j < h$, we can query the 2D range tree to get the minimum value of $T(x,y)$ such that

$1 \le x \le i$ and $j+1 \le y \le h$. Each query takes $\tilde{O}(1)$ time. Overall, the running time for this case is $O(n^3)$. ∎

It remains to consider the case where P uses some vertex on $\pi_G(s,t)$ between e_1 and e_2. We again show that there is an $O(n^3)$ time algorithm for it.

Lemma 13. *In $O(n^3)$ time, we can compute the replacement path distances $d_{G\setminus\{e_1,e_2\}}(s,t)$ for every pair of $e_1, e_2 \in \pi_G(s,t)$ where $e_1 = (p_i, p_{i+1})$ and $e_2 = (p_j, p_{j+1})$ for some $1 \le i < j < h$ and the replacement path uses some vertex p_k for $i < k \le j$.*

Proof: First we fix some e_1, e_2 and consider it's corresponding replacement path P. Without loss of generality, we assume P is canonical. Let k be the largest integer where $i < k \le j$ and P contains p_k. Also, let k' be the smallest integer where $i < k' \le k$ and the subpath of P from p_k to t (including p_k and t) uses $p_{k'}$.

Now we consider three subpaths of P separately: the subpath from s to p_k, the subpath from p_k to $p_{k'}$ and the subpath from $p_{k'}$ to t.

On the s to p_k subpath, let p_x be the rightmost vertex before e_1 and let p_y be the leftmost vertex after e_1. Since P is canonical and all edges between s and p_x do not include e_1 or e_2, the portion from s to p_x is $\pi_G(s,p_x)$, so p_y appears after p_x on the subpath. Thus, we can further decompose the s to p_k subpath to three parts: from s to p_x, from p_x to p_y and from p_y to p_k. Since P is canonical, the subpath from s to p_x and from p_y to p_k use edges entirely from $\pi_G(s,t)$, and we know these edges don't include e_1 or e_2. Thus, the lengths of these two subpaths are $d_G(s,p_x)$ and $d_G(p_y,p_k)$ respectively. We then argue that the p_x to p_y subpath cannot use any edge on $\pi_G(s,t)$. It does not use any edge between s and p_x or between p_y and p_k since that would imply P is not canonical. It does not use any edge between p_x and p_y by definitions of p_x and p_y. It does not use any edge between p_k and p_j by definition of k. Finally, it does not use any edge between p_{j+1} and t because if it does, a canonical path P should go directly to t from that edge instead of going back to p_y. Thus, the subpath from p_x to p_y has length $d_{G\setminus\pi_G(s,t)}(p_x,p_y)$. By the above discussion, the length of the subpath from s to p_k can be expressed as

$$\min_{\substack{1\le x\le i\\ i+1\le y\le k}} \left\{d_G(s,p_x) + d_{G\setminus\pi_G(s,t)}(p_x,p_y) + d_G(p_y,p_k)\right\}.$$

We can denote this value by $g_s(e_1,p_k)$, and by using 2D range tree, we can compute $g_s(e_1,p_k)$ for all values of e_1 and p_k in $\tilde{O}(n^2)$ time after computing APSP of $G \setminus \pi_G(s,t)$ in $O(n^3)$ time. More specifically, since

$$\begin{aligned} g_s(e_1,p_k) &= \min_{\substack{1\le x\le i\\ i+1\le y\le k}} \left\{d_G(s,p_x) + d_{G\setminus\pi_G(s,t)}(p_x,p_y)\right.\\ &\quad \left.+(d_G(s,p_k) - d_G(s,p_y))\right\}\\ &= d_G(s,p_k) + \min_{\substack{1\le x\le i\\ i+1\le y\le k}} \{d_G(s,p_x)\\ &\quad +d_{G\setminus\pi_G(s,t)}(p_x,p_y) - d_G(s,p_y)\}, \end{aligned}$$

we can create a table T where $T(x,y) = d_G(s,p_x) + d_{G\setminus\pi_G(s,t)}(p_x,p_y) - d_G(s,p_y)$ and store it in a 2D range tree, and then computing each $g_s(e_1,p_k)$ essentially costs a 2D range minimum query.

The subpath from $p_{k'}$ to t is similar. On the $p_{k'}$ to t subpath, let $p_{x'}$ be the rightmost vertex before e_2 and let $p_{y'}$ be the leftmost vertex after e_2. Since P is canonical, we can decompose the subpath from $p_{k'}$ to t to three subpaths: from $p_{k'}$ to $p_{x'}$, from $p_{x'}$ to $p_{y'}$ and from $p_{y'}$ to t. Since P is canonical, the subpath from $p_{k'}$ to $p_{x'}$ and the subpath from $p_{y'}$ to t have lengths $d_G(p_{k'},p_{x'})$ and $d_G(p_{y'},t)$ respectively. Now we consider the portion from $p_{x'}$ to $p_{y'}$. It cannot use any edge before p_i or after $p_{y'}$ since P is canonical. It cannot use any edge between p_{i+1} and $p_{k'}$ by definition of $p_{k'}$. It cannot use any edge between $p_{k'}$ and $p_{x'}$ since P is canonical. It cannot use any edge between $p_{x'}$ and $p_{y'}$ by definitions of $p_{x'}$ and $p_{y'}$. Thus, it entirely avoids $\pi_G(s,t)$ and its length should be $d_{G\setminus\pi_G(s,t)}(p_{x'},p_{y'})$. Therefore, the length of the subpath from $p_{k'}$ to t can be expressed as

$$\min_{\substack{k'\le x'\le j\\ j+1\le y'\le h}} \left\{d_G(p_{k'},p_{x'}) + d_{G\setminus\pi_G(s,t)}(p_{x'},p_{y'}) + d_G(p_{y'},t)\right\}.$$

We denote this value by $g_t(e_2,p_{k'})$. By using 2D range tree, we can compute $g_t(e_2,p_{k'})$ for all values of e_2 and $p_{k'}$ in $\tilde{O}(n^2)$ time after computing APSP of $G \setminus \pi_G(s,t)$ in $O(n^3)$ time. We omit the details for the 2D range tree in this case since it is almost identical to the s to p_k subpath case.

Finally, we consider the p_k to $p_{k'}$ subpath. It does not use any edge on $\pi_G(s,t)$ before e_1 or after e_2 because P is canonical. It does not use any edge after e_1 and before $p_{k'}$ or any edge after p_k and before e_2 by definitions of k and k'. Therefore, this subpath lies entirely in $G\setminus\pi_G(s,p_{k'})\setminus\pi_G(p_k,t)$. Thus, the length of this subpath is exactly $d_{G\setminus\pi_G(s,p_{k'})\setminus\pi_G(p_k,t)}(p_k,p_{k'})$, which was denoted by $f(p_k,p_{k'})$ in Section III. All values of $f(p_k,p_{k'})$ for any p_k and $p_{k'}$ can be computed deterministically in $O(n^3)$ time by Lemma 7.

Therefore, for any $e_1 = (p_i,p_{i+1})$ and $e_2 = (p_j,p_{j+1})$ for some $1 \le i < j < h$,

$$\begin{aligned} &d_{G\setminus\{e_1,e_2\}}(s,t)\\ &= \min_{i+1\le k'\le k\le j} \{g_s(e_1,p_k) + f(p_k,p_{k'}) + g_t(e_2,p_{k'})\}, \end{aligned}$$

as long as the replacement path uses some vertex on $\pi_G(s,t)$ between e_1 and e_2. To compute the right hand side of the above equation efficiently, we first create an array $A_{k,e_2}(k') = f(p_k,p_{k'}) + g_t(e_2,p_{k'})$ for every k and e_2 and build a data structure that supports range minimum queries for each array. Then for every $e_1 = (p_i,p_{i+1}), e_2 = (p_j,p_{j+1})$, we enumerate $k \in [i+1,j]$. We can write

$$\min_{i+1\le k'\le k} \{g_s(e_1,p_k) + f(p_k,p_{k'}) + g_t(e_2,p_{k'})\}$$

as

$$g_s(e_1,p_k) + \min_{i+1\le k'\le k} A_{k,e_2}(k').$$

Thus, it essentially costs one range minimum query for every triple of e_1, e_2, k. If we use range minimum query data

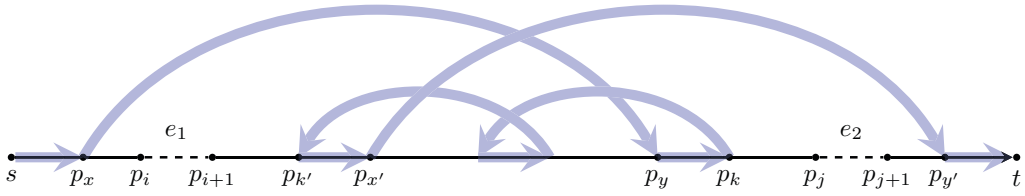

Fig. 3. This figure depicts the vertex labels for Lemma 13.

structures that supports linear time pre-processing and $O(1)$ range minimum queries (see e.g. [46]), this step takes $O(n^3)$ time.

Therefore, the overall running time is $O(n^3)$. ∎

C. Putting It All Together

Recall our Theorem 1 is the following:

Theorem 1. *In the Word-RAM Model with $O(\log n)$-bit words, the 2FRP problem on n-vertex $O(\log n)$-bit integer weighted directed graphs with no negative cycles can be solved in $O(n^3 \log^2 n)$ time by a deterministic algorithm or in $O(n^3 \log n)$ time by a randomized algorithm that succeeds with high probability.*

Proof: All components in our algorithm run in $O(n^3)$ time deterministically except the pre-processing phase of the distance sensitivity oracle from Theorem 10. Since the DSO has an $O(n^3 \log n)$ randomized pre-processing time or an $O(n^3 \log^2 n)$ deterministic pre-processing time, our algorithm for 2FRP has $O(n^3 \log n)$ randomized time or $O(n^3 \log^2 n)$ deterministic time. ∎

Using Theorem 1, we immediately obtain Corollary 2.

Corollary 2. *For all $f \geq 2$, fFRP in n-vertex directed weighted graphs with no negative cycles can be solved in $\tilde{O}(n^{f+1})$ time.*

Proof: Since the graph has no negative cycles, we can first use an $O(n^3)$ pre-processing step that replaces all edges with nonnegative edges [44], [45]. Then we compute a shortest path P_1 from s to t in $\tilde{O}(n^2)$ time using Dijkstra's algorithm. For every edge e_1 on P_1, we compute a shortest s-t path P_2 from s to t in $G \setminus \{e_1\}$. More generally, for each $i \leq f-2$, and each choice of $(e_1, \ldots, e_i)$ and computed paths $P_1, \ldots, P_i$ where each P_j is a shortest s-t path in $G \setminus \{e_1, \ldots, e_{j-1}\}$ and $e_j \in P_j$, we compute a shortest s-t path P_{i+1} in $G \setminus \{e_1, \ldots, e_i\}$. This computation takes $\tilde{O}(n^f)$ time. Then for each of the $O(n^{f-2})$ choices of $(e_1, \ldots, e_{f-2})$, we compute 2FRP using Theorem 1 in $G \setminus \{e_1, \ldots, e_i\}$ in overall time $\tilde{O}(n^{f+1})$. ∎

V. Subcubic Time Algorithm for Graphs with Bounded Integer Weights

In this section, we show how to improve the running time of 2FRP when we restrict the graphs to graphs with small integer edge weights and no negative cycles, providing proofs for Theorem 3 and Corollary 4.

A. General Approach and Intuitions

We first give some intuitions and high-level ideas of our algorithm.

Let $G = (V, E)$ be an n-vertex directed graph with integer edge weights in $\{-M \ldots M\}$ and no negative cycles. Let s be the source and t be the target for our 2FRP instance. The general approach to our algorithm is to divide $\pi_G(s,t)$ into intervals of g vertices each for a positive integer parameter $g = O(n)$.[5] Let I be one of the intervals, then we use $V(I)$ to denote the vertices inside the interval, and $E(I)$ to denote the edges inside the interval. The intervals are created in such a way that the last vertex in the previous interval is the first vertex in the next interval. Once we have created these intervals we can classify all the two-fault replacement paths queries to the following three cases: (1) only one failed edge is on $\pi_G(s,t)$, (2) both failed edges are on $\pi_G(s,t)$ in the same interval, and (3) both failed edges are on $\pi_G(s,t)$ in different intervals. Note that we don't need to consider cases where neither of the failed edges is on $\pi_G(s,t)$ as the original shortest path will exist in $G \setminus \{e_1, e_2\}$. Now, we can create three separate sub-algorithms that handle each of these cases, and combine them to get the overall 2FRP algorithm.

We will have a general precomputation step and some sub-algorithms will also have their own precomputation steps to compute any needed information that was not computed in the general precomputation step. Our approach to querying the length of the replacement path in all of the sub-algorithms is to construct a weighted auxiliary graph to aid with the query. To build one such auxiliary graph, we first determine a set of critical vertices that break down the replacement path into a series of subpaths between them. These vertices will form the vertex set of the auxiliary graph, and the edges in the auxiliary graph will represent subpaths between these vertices.

We say that the auxiliary graph *encodes* a subpath from u to v in $G \setminus \{e_1, e_2\}$ if there is a path from u to v in the auxiliary graph with the same length as the subpath. We also say that an edge (u, v) in the auxiliary graph *encodes* a subpath from u to v in $G \setminus \{e_1, e_2\}$ if the weight of that edge equals the length of the subpath. We will show many of those subpaths are encoded in the auxiliary graph, and eventually, show that

[5]For instance, we will set $g = n^{(\omega-1)/3} = O(n)$ when $M = O(1)$.

the s-t shortest path in $G \setminus \{e_1, e_2\}$ is encoded. Thus, we can run a Single-Source Shortest Paths (SSSP) algorithm on the auxiliary graph to get the length of the shortest replacement path.

B. Precomputed Distances

In the general precomputation step and the precomputation steps specific to each sub-algorithm, we will use the SSRP algorithm from [13] that has the same running time as Zwick's APSP algorithm for graphs with edge weights in $\{-M, \ldots, M\}$ and our algorithm for SSRP with a small set of targets from Lemma 9. In the precomputation steps and the query step, we will also use the near-linear time SSSP algorithm by Bernstein, Nanongkai and Wulff-Nilsen [47]. On n-vertex dense graphs, their algorithm runs in $\tilde{O}(n^2)$ time.

In this general precomputation step we compute sets of distances that will be needed for all three sub-algorithms:

1) Run SSSP and SSRP from s in G, and store the results.
2) Run SSSP and SSRP from t in $\widehat{G}$, where $\widehat{G}$ represents G with the directions of all of its edges reversed, and store the results.
3) For each interval I, create the graphs $G \setminus E(I)$ and $\widehat{G \setminus E(I)}$, then:
   a) Run SSSP and SSRP with target set $V(I) \cup \{t\}$ from s in $G \setminus E(I)$, and store the results.
   b) Run SSSP and SSRP with target set $V(I) \cup \{s\}$ from t in $\widehat{G \setminus E(I)}$ and store the results.
4) Run Zwick's All-Pairs Shortest Paths algorithm [32] on the graph with all the edges of $\pi_G(s,t)$ removed and store the results.

Overall, steps 1, 2 and 4 take $\tilde{O}(M^{1/(4-\omega)}n^{2+1/(4-\omega)})$ time, and each iteration of step 3 takes $\tilde{O}(Mn^{\omega} + M^{1/(4-\omega)}n^{1+1/(4-\omega)}g)$ time, so these pre-processing steps take $\tilde{O}(Mn^{\omega+1}/g + M^{1/(4-\omega)}n^{2+1/(4-\omega)})$ time. The space complexity of the stored results in this step is $\tilde{O}(n^2)$.

C. Only One Failed Edge on Original Shortest Path

First, we consider the case where only one of the failed edges is on $\pi_G(s,t)$. Let e_1 be the failed edge on $\pi_G(s,t)$, I_1 be the interval containing e_1, and e_2 be the failed edge that is not on $\pi_G(s,t)$. Our auxiliary graph requires distances not computed in the general precomputation step of the algorithm, so we will have a precomputation step for this algorithm. During this precomputation step, we compute a single-fault DSO for $G \setminus \pi_G(s,t)$. Using Chechik and Cohen's DSO [38], the pre-processing time is $\tilde{O}(Mn^{2.8729})$ and the space is $\tilde{O}(n^{2.5})$. If all edge weights are positive integers in $\{1, \ldots, M\}$, we can instead use Gu and Ren's DSO [25], which has $\tilde{O}(Mn^{2.5794})$ pre-processing time and $\tilde{O}(n^2)$ size. Note that although Gu and Ren's DSO has $\tilde{O}(n^2)$ size, it could use $\tilde{O}(n^{2.4207})$ space during pre-processing [25].

Let G' be the auxiliary graph, and let its vertex set be $\{s,t\} \cup V(I_1)$. We will add edges in the following steps:

1) Add an edge from s to t with weight $d_{G \setminus E(I_1)}(s,t,e_2)$.
2) For every $v \in V(I)$, add an edge (s,v) with weight $d_{G \setminus E(I_1)}(s,v,e_2)$.
3) For every $v \in V(I)$, add an edge (v,t) with weight $d_{G \setminus E(I_1)}(v,t,e_2)$.
4) Add all of the edges in I_1 that are not one of the two failed edges. Then, for every $u, v \in V(I_1)$, add the edge (u,v) with weight $d_{G \setminus \pi_G(s,t)}(u,v,e_2)$.

Now we can run SSSP from s in G'. Since there are $O(g)$ vertices in each interval, there are $O(g)$ vertices in G', so building G' and running the query takes $\tilde{O}(g^2)$ times.

Theorem 14. *$d_{G'}(s,t)$ is equal to $d_{G \setminus \{e_1,e_2\}}(s,t)$.*

Proof: Here are two main cases for the shortest s-t path that avoids e_1 and e_2: (1) the path does not use any edge in $E(I_1)$ or (2) the path does use edges in $E(I_1)$. The path for the first case is encoded in G' via the edge added in step 1.

For the second case, let P be a canonical replacement path. We know that the shortest path P must use a vertex in $V(I_1)$. Let u be the first vertex on the replacement path that is in $V(I_1)$, and v be the last vertex on the replacement path that is in $V(I_1)$. Then, the replacement path can be broken down into three subpaths: (1) a subpath from s to u, (2) a subpath from u to v, and (3) a subpath from v to t. If G' encodes each of these subpaths for every possible value of u and v, and the replacement path does use edges in $E(I_1)$, then $\pi_{G'}(s,t)$ will use the optimal choices for u and v, which will give us the length of the replacement path.

First, we will focus on the subpaths from s to u for all choices of u. Since u will be the first vertex in $V(I_1)$ on the replacement path, this subpath will not use any edge in $E(I_1)$. Similarly, the subpaths from v to t will not use any edge in $E(I_1)$ either, since v is the last vertex in $V(I_1)$ on the replacement path. All of the subpaths must also avoid e_2, since it is a failed edge. Therefore, the edges added in steps 2 and 3 are sufficient to encode the s-u subpaths and v-t subpaths into G'.

Next, we will focus on the subpaths between u and v for all choices of u and v. If a canonical replacement path does travel between two vertices in $V(I_1)$, then it will not use any edge on $\pi_G(s,t)$ outside of I_1 to do so, as that would prevent it from being canonical. For example, if the path between u and v reached a vertex w in $\pi_G(s,t)$ before I_1, then the replacement path should go directly from s to w, instead of going to u first, because it is canonical. The mirror situation occurs if it touches a vertex after I_1 and cannot happen for similar reasons. Therefore, when traveling between two vertices in $V(I_1)$, the replacement path will only use edges in $E(I_1)$ and edges not on $\pi_G(s,t)$. As a result, every u-v subpath can be broken into a series of smaller paths consisting of edges in $E(I_1)$ and paths between vertices in $V(I_1)$ that do not use any edge on $\pi_G(s,t)$. The edges in step 4 encode all of these smaller paths into G', and as a result every u-v subpath is encoded in G'.

In total, G' encodes every possible subpath which the replacement path could be constructed from, so the shortest s-t path in G' can not be longer than the replacement path in $G \setminus \{e_1, e_2\}$. It is impossible for $d_{G'}(s,t)$ to be smaller

than $d_{G\setminus\{e_1,e_2\}}(s,t)$ because all of the edges in G' have weights that correspond to the lengths of some paths that are present in $G\setminus\{e_1,e_2\}$. Therefore, $d_{G'}(s,t)$ must be equal to $d_{G\setminus\{e_1,e_2\}}(s,t)$. ∎

D. Both Failed Edges on Original Shortest Path: Same Interval

Next, we will consider the case where both failed edges e_1, e_2 are on $\pi_G(s,t)$ in the same interval I. We start by constructing the auxiliary graph for this query. Let G' be the auxiliary graph, and let its vertex set be $\{s,t\}\cup V(I)$. We will add edges in the following steps:

1) Add an edge from s to t with weight $d_{G\setminus E(I)}(s,t)$.
2) For every $v \in V(I)$, add an edge (s,v) with weight $d_{G\setminus E(I)}(s,v)$.
3) For every $v \in V(I)$, add an edge (v,t) with weight $d_{G\setminus E(I)}(v,t)$.
4) Add all of the edges in I that are not one of the two failed edges. Then, for every $u,v \in V(I)$, add an edge (u,v) with weight $d_{G\setminus\pi_G(s,t)}(u,v)$.

Now we can run SSSP from s in G'. Since there are $O(g)$ vertices in each interval, there are $O(g)$ vertices in G', so building G' and running the query takes $\tilde{O}(g^2)$ time. All of the edge weights of G' were already calculated in the general precomputation step, so there is no additional precomputation step for this sub-algorithm.

Theorem 15. *$d_{G'}(s,t)$ is equal to $d_{G\setminus\{e_1,e_2\}}(s,t)$.*

The proof for this case is similar to the proof of Theorem 14. We defer its proof to the full version.

E. Both Failed Edges on Original Shortest Path: Different Intervals

Due to space limitation, we will defer this case to the full version. The precomputation step of this case takes $\tilde{O}(Mn^{\omega+1}/g + M^{1/(4-\omega)}n^{2+1/(4-\omega)} + M^{1/3}n^{2+\omega/3})$ time. We can improve the third term to $\tilde{O}(M^{0.3544}n^{2.7778})$ using rectangular matrix multiplication. The query time is $\tilde{O}(g^2)$ and the space complexity is again $\tilde{O}(n^2)$.

F. Putting It All Together

Now we have all the necessary components for proving Theorem 3, which is recalled here:

Theorem 3. *For any given positive integer parameter $g \le O(n)$, there exists a data structure that can pre-process a given directed graph G with integer edge weights in $\{-M,\dots,M\}$ and no negative cycles and fixed vertices s and t, in $\tilde{O}(Mn^{\omega+1}/g + Mn^{2.8729})$ time, and can answer queries of the form $d_{G\setminus\{e_1,e_2\}}(s,t)$ in $\tilde{O}(g^2)$ time. This data structure has randomized pre-processing which succeeds with high probability. The size of the data structure is $\tilde{O}(n^{2.5})$.*

If the edge weights of G are positive, the pre-processing time and the size of the data structure can be improved to $\tilde{O}(Mn^{\omega+1}/g + M^{0.3544}n^{2.7778} + Mn^{2.5794})$ and $\tilde{O}(n^2)$ respectively.

Proof: In total, the running time for the general precomputation phase and the precomputation phase of each sub-algorithm is $\tilde{O}(Mn^{\omega+1}/g + M^{1/(4-\omega)}n^{2+1/(4-\omega)} + M^{1/3}n^{2+\omega/3} + Mn^{2.8729})$. The fourth term dominates the second and third term, so the pre-processing time simplifies to $\tilde{O}(Mn^{\omega+1}/g + Mn^{2.8729})$. The space complexity is $\tilde{O}(n^{2.5})$, where the space of the DSO is the bottleneck.

If all edge weights are positive, the pre-processing time can be improved to $\tilde{O}(Mn^{\omega+1}/g + M^{1/(4-\omega)}n^{2+1/(4-\omega)} + M^{1/3}n^{2+\omega/3} + Mn^{2.5794})$. The third term can be improved to $\tilde{O}(M^{0.3544}n^{2.7778})$ using rectangular matrix multiplication by Lemma 8. Note that the second term is always dominated by the third term or the fourth term, so the pre-processing time simplifies to $\tilde{O}(Mn^{\omega+1}/g + M^{0.3544}n^{2.7778} + Mn^{2.5794})$. The space complexity is $\tilde{O}(n^2)$.

The query time is $\tilde{O}(g^2)$ in both cases, since we always run the near-linear time SSSP algorithm [47] on an auxiliary graph with $O(g)$ vertices. ∎

We can easily obtain our algorithm for 2FRP from Theorem 3 by setting the parameter appropriately. Recall Corollary 4:

Corollary 4. *The 2FRP problem on n-vertex directed graphs with integer edge weights in $\{-M,\dots,M\}$ and with no negative cycles can be solved in $\tilde{O}(M^{2/3}n^{2.9153})$ time by a randomized algorithm that succeeds with high probability.*

Proof: We run the near-linear time SSSP algorithm [47] to find $\pi_G(s,t)$ in $\tilde{O}(n^2)$ time, and then run the $\tilde{O}(Mn^{\omega})$ time RP algorithm by Vassilevska Williams [12] to find $\pi_{G\setminus\{e\}}(s,t)$ for every $e \in \pi_G(s,t)$.

Then we can easily generate all (e_1,e_2) pairs such that $e_1 \in \pi_G(s,t)$ and $e_2 \in \pi_{G\setminus\{e_1\}}(s,t)$ in $O(n^2)$ time. Using Theorem 3, it will take $\tilde{O}(g^2n^2 + Mn^{\omega+1}/g + Mn^{2.8729})$ time to handle all the queries, which is

$$\tilde{O}(M^{2/3}n^{2(\omega+2)/3} + Mn^{2.8729})$$

by setting $g = M^{1/3}n^{(\omega-1)/3}$. Using the current upper bound $\omega < 2.3729$, the running time becomes $\tilde{O}(M^{2/3}n^{2.9153} + Mn^{2.8729})$. Note that the second term is larger than the first term only when $M^{2/3}n^{2.9153} = \Omega(n^3)$, so we can just run the $\tilde{O}(n^3)$ time algorithm from Theorem 1 in this case. Thus, the running time is always $\tilde{O}(\min(M^{2/3}n^{2.9153}, n^3)) = \tilde{O}(M^{2/3}n^{2.9153})$.

Finally, we need to check $g = O(n)$ by the requirement of Theorem 3. Note that this value of g is $O(n)$ when $M = O(n^{4-\omega})$. When $M = \Omega(n^{4-\omega})$, our claimed running time exceeds $\Omega(n^3)$ and thus we can just run the $\tilde{O}(n^3)$ time algorithm from Theorem 1. ∎

REFERENCES

[1] K. Malik, A. K. Mittal, and S. K. Gupta, "The k most vital arcs in the shortest path problem," *Operations Research Letters*, vol. 8, no. 4, pp. 223–227, 1989.

[2] E. Nardelli, G. Proietti, and P. Widmayer, "A faster computation of the most vital edge of a shortest path," *Information Processing Letters*, vol. 79, no. 2, pp. 81–85, 2001.

[3] Z. Gotthilf and M. Lewenstein, "Improved algorithms for the k simple shortest paths and the replacement paths problems," *Information Processing Letters*, vol. 109, no. 7, pp. 352–355, 2009.

[4] A. Bernstein, "A nearly optimal algorithm for approximating replacement paths and k shortest simple paths in general graphs," in *Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010, pp. 742–755.

[5] V. Vassilevska Williams and R. Williams, "Subcubic equivalences between path, matrix, and triangle problems," *J. ACM*, vol. 65, no. 5, pp. 1–38, 2018.

[6] Y. Emek, D. Peleg, and L. Roditty, "A near-linear-time algorithm for computing replacement paths in planar directed graphs," *ACM Trans. Algorithms*, vol. 6, no. 4, pp. 1–13, 2010.

[7] P. N. Klein, S. Mozes, and O. Weimann, "Shortest paths in directed planar graphs with negative lengths: A linear-space $O(n \log^2 n)$-time algorithm," *ACM Trans. Algorithms*, vol. 6, no. 2, pp. 1–18, 2010.

[8] A. M. Bhosle, "Improved algorithms for replacement paths problems in restricted graphs," *Oper. Res. Lett.*, vol. 33, no. 5, pp. 459–466, 2005.

[9] C.-W. Lee and H.-I. Lu, "Replacement paths via row minima of concise matrices," *SIAM J. Discrete Math.*, vol. 28, no. 1, pp. 206–225, 2014.

[10] L. Roditty and U. Zwick, "Replacement paths and k simple shortest paths in unweighted directed graphs," *ACM Trans. Algorithms*, vol. 8, no. 4, Oct. 2012. [Online]. Available: https://doi.org/10.1145/2344422.2344423

[11] O. Weimann and R. Yuster, "Replacement paths via fast matrix multiplication," in *Proc. 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2010, pp. 655–662.

[12] V. Vassilevska Williams, "Faster replacement paths," in *Proc. 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2011, pp. 1337–1346.

[13] F. Grandoni and V. Vassilevska Williams, "Faster replacement paths and distance sensitivity oracles," *ACM Trans. Algorithms*, vol. 16, no. 1, pp. 15:1–15:25, Dec. 2020. [Online]. Available: https://doi.org/10.1145/3365835

[14] N. Nisan and A. Ronen, "Algorithmic mechanism design," *Games Econ. Behav.*, vol. 35, no. 1-2, pp. 166–196, 2001.

[15] J. Hershberger and S. Suri, "Vickrey prices and shortest paths: What is an edge worth?" in *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, 2001, pp. 252–259.

[16] V. Vassilevska Williams and R. Williams, "Subcubic equivalences between path, matrix and triangle problems," in *Proc. 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2010, pp. 645–654.

[17] A. M. Bhosle and T. F. Gonzalez, "Replacement paths for pairs of shortest path edges in directed graphs," in *Proc. 16th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS)*, 2004.

[18] J. Alman and V. Vassilevska Williams, "A refined laser method and faster matrix multiplication," in *Proc. 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2021, pp. 522–539.

[19] F. Le Gall, "Powers of tensors and fast matrix multiplication," in *Proc. 39th International Symposium on Symbolic and Algebraic Computation (ISSAC)*, 2014, pp. 296–303.

[20] V. Vassilevska Williams, "Multiplying matrices faster than Coppersmith-Winograd," in *Proc. 44th Annual ACM Symposium on Theory of Computing (STOC)*, 2012, pp. 887–898.

[21] Y. Gu, A. Polak, V. Vassilevska Williams, and Y. Xu, "Faster Monotone Min-Plus Product, Range Mode, and Single Source Replacement Paths," in *Proc. 48th International Colloquium on Automata, Languages, and Programming (ICALP)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 198, 2021, pp. 75:1–75:20. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2021/14144

[22] O. Weimann and R. Yuster, "Replacement paths and distance sensitivity oracles via fast matrix multiplication," *ACM Trans. Algorithms*, vol. 9, no. 2, Mar. 2013. [Online]. Available: https://doi.org/10.1145/2438645.2438646

[23] J. van den Brand and T. Saranurak, "Sensitive distance and reachability oracles for large batch updates," in *Proc. 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2019, pp. 424–435.

[24] F. Le Gall and F. Urrutia, "Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor," in *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2018, pp. 1029–1046.

[25] Y. Gu and H. Ren, "Constructing a Distance Sensitivity Oracle in $O(n^{2.5794}M)$ Time," in *Proc. 48th International Colloquium on Automata, Languages, and Programming (ICALP)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 198, 2021, pp. 76:1–76:20. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2021/14145

[26] M. Gupta and S. Khan, "Multiple source dual fault tolerant BFS trees," in *Proc. 44th International Colloquium on Automata, Languages, and Programming (ICALP)*, ser. LIPIcs, vol. 80, 2017, pp. 127:1–127:15.

[27] M. Parter and D. Peleg, "Sparse fault-tolerant BFS structures," *ACM Trans. Algorithms*, vol. 13, no. 1, pp. 11:1–11:24, 2016.

[28] R. Williams, "Faster all-pairs shortest paths via circuit complexity," in *Proc. 46th Annual ACM Symposium on Theory of Computing (STOC)*, 2014, pp. 664–673.

[29] R. R. Williams, "Faster all-pairs shortest paths via circuit complexity," *SIAM J. Comput.*, vol. 47, no. 5, pp. 1965–1985, 2018.

[30] R. Seidel, "On the all-pairs-shortest-path problem in unweighted undirected graphs," *J. Comput. Syst. Sci*, vol. 51, no. 3, pp. 400–403, 1995.

[31] A. Shoshan and U. Zwick, "All pairs shortest paths in undirected graphs with integer weights," in *Proc. 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 1999, pp. 605–614.

[32] U. Zwick, "All pairs shortest paths using bridging sets and rectangular matrix multiplication," *J. ACM*, vol. 49, no. 3, pp. 289–317, 2002.

[33] N. Alon, S. Chechik, and S. Cohen, "Deterministic combinatorial replacement paths and distance sensitivity oracles," in *Proc. 46th International Colloquium on Automata, Languages, and Programming (ICALP)*, 2019.

[34] F. Grandoni and V. Vassilevska Williams, "Improved distance sensitivity oracles via fast single-source replacement paths," in *Proc. 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2012, pp. 748–757.

[35] S. Chechik and O. Magen, "Near Optimal Algorithm for the Directed Single Source Replacement Paths Problem," in *Proc. 47th International Colloquium on Automata, Languages, and Programming (ICALP)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 168, 2020, pp. 81:1–81:17. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2020/12488

[36] C. Demetrescu, M. Thorup, R. A. Chowdhury, and V. Ramachandran, "Oracles for distances avoiding a failed node or link," *SIAM J. Comput.*, vol. 37, no. 5, pp. 1299–1318, 2008.

[37] A. Bernstein and D. Karger, "A nearly optimal oracle for avoiding failed vertices and edges," in *Proc. 41st Annual ACM Symposium on Theory of Computing (STOC)*, 2009, pp. 101–110.

[38] S. Chechik and S. Cohen, "Distance sensitivity oracles with subcubic preprocessing time and fast query time," in *Proc. 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2020, pp. 1375–1388.

[39] H. Ren, "Improved distance sensitivity oracles with subcubic preprocessing time," *J. Comput. Syst. Sci.*, vol. 123, pp. 159–170, 2022.

[40] R. Duan and S. Pettie, "Dual-failure distance and connectivity oracles," in *Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2009, pp. 506–515.

[41] R. Duan and H. Ren, "Maintaining exact distances under multiple edge failures," in *Proc. the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2022, p. 1093–1101. [Online]. Available: https://doi.org/10.1145/3519935.3520002

[42] F. Le Gall, "Faster algorithms for rectangular matrix multiplication," in *Proc. 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2012, pp. 514–523.

[43] N. Alon, Z. Galil, O. Margalit, and M. Naor, "Witnesses for boolean matrix multiplication and for shortest paths," in *Proc. 33rd Annual Symposium on Foundations of Computer Science*, 1992, pp. 417–426.

[44] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *J. ACM*, vol. 24, no. 1, pp. 1–13, 1977.

[45] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *J. ACM*, vol. 34, no. 3, pp. 596–615, 1987.

[46] D. Harel and R. E. Tarjan, "Fast algorithms for finding nearest common ancestors," *SIAM J. Comput.*, vol. 13, no. 2, pp. 338–355, 1984.

[47] A. Bernstein, D. Nanongkai, and C. Wulff-Nilsen, "Negative-weight single-source shortest paths in near-linear time," *arXiv preprint arXiv:2203.03456*, 2022.