Approximate Distance Sensitivity Oracles in Subquadratic Space

Davide Bilò
Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila
L'Aquila, Italy
davide.bilo@univaq.it

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

Keerti Choudhary
Department of Computer Science and Engineering, Indian Institute of Technology Delhi
New Delhi, India
keerti@iitd.ac.in

Sarel Cohen
School of Computer Science, The Academic College of Tel Aviv-Yaffo
Tel Aviv, Israel
sarelco@mta.ac.il

Tobias Friedrich
Simon Krogmann
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany
firstname.lastname@hpi.de

Martin Schirneck
Faculty of Computer Science, University of Vienna
Vienna, Austria
martin.schirneck@univie.ac.at

ABSTRACT

An *f-edge fault-tolerant distance sensitive oracle* (f-DSO) with stretch $\sigma \geqslant 1$ is a data structure that preprocesses a given undirected, unweighted graph G with n vertices and m edges, and a positive integer f. When queried with a pair of vertices s, t and a set F of at most f edges, it returns a σ-approximation of the s-t-distance in $G - F$.

We study f-DSOs that take subquadratic space. Thorup and Zwick [JACM 2015] showed that this is only possible for $\sigma \geqslant 3$. We present, for any constant $f \geqslant 1$ and $\alpha \in (0, \frac{1}{2})$, and any $\varepsilon > 0$, an f-DSO with stretch $3 + \varepsilon$ that takes $\widetilde{O}(n^{2-\frac{\alpha}{f+1}}/\varepsilon) \cdot O(\log n/\varepsilon)^{f+1}$ space and has an $O(n^{\alpha}/\varepsilon^2)$ query time.

We also give an improved construction for graphs with diameter at most D. For any constant k, we devise an f-DSO with stretch $2k - 1$ that takes $O(D^{f+o(1)}n^{1+1/k})$ space and has $\widetilde{O}(D^{o(1)})$ query time, with a preprocessing time of $O(D^{f+o(1)}mn^{1/k})$.

Chechik, Cohen, Fiat, and Kaplan [SODA 2017] presented an f-DSO with stretch $1+\varepsilon$ and preprocessing time $\widetilde{O}_{\varepsilon}(n^5)$, albeit with a super-quadratic space requirement. We show how to reduce their preprocessing time to $O(mn^2) \cdot O(\log n/\varepsilon)^f$.

CCS CONCEPTS

• **Theory of computation** → **Data structures design and analysis**; *Shortest paths*; • **Mathematics of computing** → *Graph algorithms.*

KEYWORDS

approximate shortest paths, distance sensitivity oracle, fault-tolerant data structure, subquadratic space

STOC '23, June 20–23, 2023, Orlando, FL, USA

ACM ISBN 978-1-4503-9913-5/23/06…$15.00
https://doi.org/10.1145/3564246.3585251

ACM Reference Format:
Davide Bilò, Shiri Chechik, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, Simon Krogmann, and Martin Schirneck. 2023. Approximate Distance Sensitivity Oracles in Subquadratic Space. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC '23), June 20–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3564246.3585251

1 INTRODUCTION

Distance Oracles (DOs) are fundamental data structures that store information about the distances of an input graph $G = (V, E)$.[1] These oracles are used in several applications where one cannot afford to store the entire input, but still wants to quickly retrieve the graph distances upon query. Therefore, DOs should provide reasonable trade-offs between space consumption, query time, and *stretch*, that is, the quality of the estimated distance.

We are interested in the design of DOs that additionally can tolerate multiple failures of edges in G. An *f-edge fault-tolerant distance sensitivity oracles* (f-DSO) is able to report an estimate $\widehat{d}_{G-F}(s,t)$ of the distance $d_{G-F}(s,t)$ between s and t in the graph $G - F$, where $F \subseteq E$ is a set of at most f failing edges, when queried with the triple (s, t, F). The parameter f is the *sensitivity* of the DSO. We say that the *stretch* of the f-DSO is $\sigma \geqslant 1$ if $d_{G-F}(s,t) \leqslant \widehat{d}_{G-F}(s,t) \leqslant \sigma \cdot d_{G-F}(s,t)$ holds for every query (s, t, F).

Several f-DSOs with different size-stretch-time trade-offs have been proposed in the last decades, some of which can only deal with a very small number $f \in \{1, 2\}$ of failures [3, 5, 6, 8, 13, 17–19, 22, 23, 27]. In the following, we focus on f-DSOs that deal with multiple failures $f \geqslant 3$. The Monte Carlo f-DSO of Weimann and Yuster [30] computes exact distances w.h.p.[2] and gives adjustable trade-offs depending on some parameter $\alpha \in (0, 1)$. More precisely, the f-DSO can be built in $\widetilde{O}(mn^{2-\alpha})$ time, has a query time of $\widetilde{O}(n^{2-2(1-\alpha)/f})$, and uses $\widetilde{O}(n^{3-\alpha})$ space.[3] The f-DSO of Duan and Ren [20] requires $O(fn^4)$ space, returns exact distances in $f^{O(f)}$

[1] Throughout, we assume the graph G to be undirected and unweighted. We use n for the number of vertices and m for the number of edges.
[2] An event occurs *with high probability* (w.h.p.) if it has probability at least $1 - n^{-c}$ for some constant $c > 0$.
[3] The space is measured in the number of machine words on $O(\log n)$ bits. For a function g of the input and parameters, we use $\widetilde{O}(g)$ to denote $O(g \cdot \text{polylog}(n))$.

query time, but the preprocessing algorithm that builds takes $n^{\Omega(f)}$ time. The f-DSO of Chechik, Cohen, Fiat, and Kaplan [14] can handle up to $f = o(\log n/\log\log n)$ failures but has a stretch of $1+\varepsilon$, for any constant $\varepsilon > 0$. In turn, the oracle is more compact requiring $O_\varepsilon(n^{2+o(1)}\log W)$ space, where W is the weight of the heaviest edge of G, has query time $O_\varepsilon(f^5 \log n \log\log W)$, and can be build in $\widetilde{O}_\varepsilon(n^{5+o(1)}\log W)$ preprocessing time. Note that the aforementioned f-DSOs all have a super-quadratic space requirement, that is, they take up *more* space than the original input graph, which is prohibitive in settings where we cannot even afford to store G. The f-DSO of Chechik, Langberg, Peleg, and Roditty [16] addresses this issue with a space requirement of $O(fkn^{1+1/k}\log(nW))$, where $k \geqslant 1$ is an integer parameter. Their data structure has a fast query time of $\widetilde{O}(|F|\log\log d_{G-F}(s,t))$ but guarantees only a stretch of $(8k-2)(f+1)$, that is, depending on the sensitivity f.

Another way to provide approximate pairwise replacement distances under edge failures is that of fault-tolerant spanners [26]. An *(f-edge) fault-tolerant σ-spanner* is a subgraph H of G such that $d_{H-F}(s,t) \leqslant \sigma \cdot d_{G-F}(s,t)$, for every triple (s,t,F), with $s,t \in V$ and $F \subseteq E$, $|F| \leqslant f$. There is a simple algorithm by Chechik, Langberg, Peleg, and Roditty [15] that computes, for any positive integer k, a fault-tolerant $(2k-1)$-spanner with $O(fn^{1+1/k})$ edges. Constructions by Bodwin, Dinitz, and Robelle [10, 11] recently reduced the size to $f^{1/2}n^{1+1/k}\cdot\mathrm{poly}(k)$ for even k, and $f^{1/2-1/(2k)}n^{1+1/k}\cdot\mathrm{poly}(k)$ for odd k. They also showed an almost matching lower bound of $\Omega(f^{1/2-1/(2k)}n^{1+1/k}+fn)$ for $k>2$, and $\Omega(f^{1/2}n^{3/2})$ for $k=2$, assuming the Erdős girth conjecture [21]. The space is also the main problem with this approach as it translates to a high query time. Currently, the most efficient way to retrieve the approximate distance between a given pair of vertices is to compute the single-source distance from one of them in time that is at least linear in the size of the spanner.

All the results above for multiple failures either require $\Omega(n^2)$ space, have a stretch depending on f, or superlinear query time. If we want a truly constant stretch for every (constant) sensitivity f and fast query time simultaneously, one currently has to pay $\Omega(n^2)$ space. A natural question is whether we can break the quadratic barrier. In this paper, we answer this question affirmatively.

Theorem 1.1. *Let $f \geqslant 2$ be a positive integer and $0 < \alpha < 1/2$ a constant. For any undirected, unweighted graph G with unique shortest paths and any $\varepsilon > 0$, there is a $(3+\varepsilon)$-approximate f-DSO for G that takes space $\widetilde{O}(n^{2-\frac{\alpha}{f+1}}/\varepsilon)\cdot O(\log n/\varepsilon)^{f+1}$, has query time $O(n^\alpha/\varepsilon^2)$, and preprocessing time $\widetilde{O}(n^{2-\frac{\alpha}{f+1}}(\frac{m}{\varepsilon}+\frac{1}{\varepsilon^2}))\cdot O(\log n/\varepsilon)^f$.*

The assumption of unique shortest paths in the base graph G can be achieved by perturbing the edge weights of the input. For unweighted graphs, this results in weighted graphs where every edge weight is very close to 1, which is a sufficient alternative condition for all the places in the paper where we assume that the graph is unweighted. Alternatively, we can compute a set of unique paths via *lexicographic pertubation* [12] in time $O(mn+n^2\log^2 n)$. To obtain Theorem 1.1, we develop several new techniques. For the remainder of this section, we highlight the novelties. A more detailed overview of our construction can be found in Section 2.

Tree Sampling for Short Paths. It is a common approach in the design of fault-tolerant data structures to first give a solution for short paths and then combine them to one for all distances, see [13, 22, 23, 25, 27, 30]. We also focus first on f-DSOs for short paths. Let L be the cut-off parameter.[4] We say a path is *short* if it has at most L edges. An f-DSO for short paths only needs to report the correct answer for a query (s,t,F) if $G-F$ contains a shortest path from s to t with at most L edges. Designing such an oracle with good query-space-preprocessing trade-offs is the first step towards improving general f-DSOs. Let $d^{\leqslant L}_{G-F}(s,t)$ be the minimum length over all s-t-paths in $G-F$ with at most L edges; if there are none, then $d^{\leqslant L}_{G-F}(s,t) = +\infty$. Note that $d^{\leqslant L}_{G-F}(s,t) = +\infty$ may hold for pairs (s,t) that are connected in $G-F$.

Theorem 1.2. *Let f, k be positive integers. There exists a data structure that, when given an undirected, unweighted graph $G=(V,E)$, and a positive integer L (possibly dependent n and m), preprocesses G and answers queries (s,t,F) for vertices $s,t \in V$ and sets of edges $F \subseteq E$ with $|F| \leqslant f$. W.h.p. over all queries, the returned value $\widehat{d^{\leqslant L}}(s,t,F)$ satisfies $d_{G-F}(s,t) \leqslant \widehat{d^{\leqslant L}}(s,t,F) \leqslant (2k-1)\cdot d^{\leqslant L}_{G-F}(s,t,F)$. The data structure takes space $\widetilde{O}(L^{f+o(1)}n^{1+1/k})$, has query time $\widetilde{O}(L^{o(1)})$, and preprocessing time $\widetilde{O}(L^{f+o(1)}mn^{1/k})$.*

We compare Theorem 1.2 with previous work on f-DSOs for short paths. Weimann and Yuster [30] presented a construction with $\widetilde{O}(L^f mn)$ preprocessing time, $\widetilde{O}(L^f n^2)$ space, and $\widetilde{O}(L^f)$ query time. It laid the foundation for many subsequent works, see [2, 7, 9, 25, 27]. When using the fault-tolerant trees described in Appendix A of [14], one can reduce the query time of the oracle to $O(f^2)$. However, storing all of these fault-tolerant trees still requires $\Omega(L^f n^2)$ space. For small enough L, sub-quadratic space suffices for our data structure, while still providing a better query time then [30].

In order to prove Theorem 1.2, we extend the sampling technique by Weimann and Yuster [30]. It consists of first constructing $\widetilde{O}(L^f)$ copies of G and then, in each one, remove edges with probability $1/L$. One can show that w.h.p. each short replacement path survives in one of the copies, where a *replacement path* is the respective shortest path after at most f edge failures. Instead of having all those graphs be independent of each other, we develop hierarchical tree sampling. This allows us to quickly find the copies that are relevant for a given query, reducing the query time to $\widetilde{O}(L^{o(1)})$. We further sparsify the resulting graphs for a better space complexity.

From Theorem 1.2, we immediately get an f-DSO for graphs with bounded diameter. Afek, Bremler-Barr, Kaplan, Cohen, and Merritt [1] proved that for undirected, unweighted graphs G any shortest path in $G-F$ is a concatenation of up to $|F|+1$ shortest paths in G. If G has diameter at most D and $|F| \leqslant f$, the diameter of $G-F$ is thus bounded by $(f+1)D$. This gives the following corollary.

Corollary 1.3. *Let f and k be positive integers. There exists a $(2k-1)$-approximate f-DSO for undirected, unweighted graphs with diameter bounded by D, that takes space $\widetilde{O}(D^{f+o(1)}n^{1+1/k})$, has query time $\widetilde{O}(D^{o(1)})$, and preprocessing time $\widetilde{O}(D^{f+o(1)}mn^{1/k})$.*

Fault-Tolerant Trees with Granularity. We employ fault-tolerant trees[5] (FT-trees) introduced by Chechik et al. [14] to combine the solutions for short paths. Those are trees in which every node is

[4] The cut-off point will eventually turn out to be $L = n^{\alpha/(f+1)}$, where $\alpha \in (0, \frac{1}{2})$ is the parameter from Theorem 1.1.

[5] FT-trees are not related to the tree sampling mentioned before.

associated with a path in a subgraph $G - A$ where $A \subseteq E$ is a set of edges, but possibly much more than the sensitivity f. Each path is partitioned into segments whose sizes increase exponentially towards the middle. This is done to encode the paths more space efficient than edge-by-edge. We have to take some additional compression steps to fit them in subquadratic space. For example, instead of building a tree $FT(s,t)$ for every pair of vertices s,t, we only do so if one of them is from a set of randomly selected *pivots*. But even this gives only a sub-linear query time. To improve it further to $\widetilde{O}_\varepsilon(n^\alpha)$ for an any constant $\alpha \in (0, \frac{1}{2})$, we generalize the FT-trees by adding what we call *granularity* $\lambda \geqslant 0$.[6] That means the first and last λ edges of each path are their own segment and do not fall into the regime of exponential increase. The original construction in [14] corresponds to granularity 0. Intuitively, the larger the value of λ, the better the fault-tolerant tree $FT_\lambda(u,v)$ with granularity λ approximates the shortest distance from u to v in $G - F$, but the larger the size of each node of the tree becomes.

The idea to answer a query (s,t,F) is to scan balls of a certain radius around s and t in $G-F$ for pivots and query the respective FT-tree together with the oracle for short paths in Theorem 1.2. W.h.p. one of the pivots hits the replacement path from s to t ensuring that this gives (an approximation of) the right distance. The bottleneck is the case when there are too many vertices in the vicinity of both s and t since then these balls also receive many pivots. Instead, we sample a second type of much more scarce pivots, which are used to hit the dense neighbourhoods. In that case, we can find a scarce pivot b_s near s and a scarce pivot b_t near t, but we can no longer assume that they hit the sought replacement path. The fault-tolerant tree $FT_\lambda(b_s,b_t)$ with granularity λ, however, allows us to get a good approximation, as long the starting points b_s and b_t are at distance at most λ from the real endpoints.

The trees $FT_\lambda(b_s,b_t)$ are much larger than their classical counterparts $FT(s,t)$. This is compensated by the fact that we require much fewer of those. We verify that several of the key lemmas from [14] transfer to fault-tolerant trees with granularity $\lambda > 0$.

Efficient Computation of Expaths. Since fault-tolerant trees are crucial for our work, we revisit the approach used by Chechik et al. [14] to construct them (with granularity 0). It turns out that their algorithm can be improved. The preprocessing in [14] invokes many calls to all-pairs shortest path computations (APSP) in different subgraphs $G-F$, each of which is associated with a node of the fault-tolerant trees. They also invoke $O(n)$ calls to Dijkstra's algorithm on suitable dense graphs with $O(fn^2)$ edges. We prove that many of those APSP calls can be avoided by instead re-using the distances in the original graph G, which can be obtained by a single APSP computation. More precisely, the paths associated with the nodes of the fault-tolerant trees (later referred as $(2f+1)$-expaths) are the concatenation of $O(f \log(nW))$ original shortest paths. The distances in G can be integrated into a single Dijkstra run on a specially built graph with $\widetilde{O}(fm)$ edges to compute such an expath in time $\widetilde{O}(fm)$. This technique implies an improved preprocessing time for our own subquadratic f-DSO. Moreover, when plugged in the preprocessing algorithm in [14], it improves the overall time complexity from $O_\varepsilon(fn^{5+o(1)})$ to $O_\varepsilon(fmn^{2+o(1)})$.

[6] In the proof of Theorem 1.1, we set $\lambda = \varepsilon L/c$, for an ad-hoc constant $c > 1$.

Due to space reasons, the details of the improved expath computation are omitted from this extended abstract.

Theorem 1.4. *Let G be an undirected weighted graph with maximum edge weight $W = \text{poly}(n)$, and unique shortest paths. For any positive integer $f = o(\log n/\log\log n)$, and $\varepsilon \geqslant 1/nW$, there exists an $(1+\varepsilon)$-approximate f-DSO for G that takes space $\widetilde{O}(fn^2)O\left(\frac{\log(nW)}{\varepsilon}\right)^f = O(fn^{2+o(1)}/\varepsilon^f)$, has query time $O(f^5\log n)$, and preprocessing time $O(fmn^2)\cdot O\left(\frac{\log(nW)}{\varepsilon}\right)^f = O(fmn^{2+o(1)}/\varepsilon^f)$.*

Open Problems. As an open question, we ask whether one can further improve the query time from $\widetilde{O}_\varepsilon(n^\alpha)$ to poly-logarithmic in n and $1/\varepsilon$ while keeping the space truly subquadratic. The converse open problem is to further reduce the space without affecting the query time. Finally, we can only handle unweighted graphs currently where the length of the path corresponds to the number of edges. Some of the sampling-based ideas break down if long paths can consist of only a few heavy edges. For all the open problems the bottleneck is the case of long paths. For short path distances we have an f-DSO of asymptotically almost optimal size and very low query time that can easily be adapted to the weighted case.

2 OVERVIEW

Fault-tolerant Trees. Our distance sensitivity oracle is built on the concept of fault-tolerant trees [14]. This is a data structure that reports, for a fixed pair of vertices $s,t \in V$ and any set $F \subseteq E$ of up to f edge failures, the replacement distance $d_{G-F}(s,t)$. Consider a shortest path P from s to t in the original graph G. FT-trees draw from the fact that only failures on P can influence the distance from s to t. In its simplest form, the tree $FT(s,t)$ consists of a root node that stores the path P and the distance $d(s,t) = |P|$. It has a child for each edge $e \in E(P)$ which in turn holds a shortest s-t-path in $G-e$. Iterating this construction until depth f ensures that all relevant failure sets for the pair (s,t) are covered. If some set of edge failures disconnect the two vertices, this is represented by a leaf node that does not store any path. Let P_v denote the path in some node v. Given a failure set F, the algorithm checks in each node v starting with the root whether it is a leaf or $F \cap E(P_v) = \emptyset$, with the latter meaning that the path P_v exists in $G-F$. If so, its length $|P_v|$ is reported; otherwise, the search recurses on the child node corresponding to an (arbitrary) edge $e \in F \cap E(P_v)$. Let $FT(s,t,F)$ be the reported distance. It is equal to $d_{G-F}(s,t)$ and the query time is $O(f^2)$ since at most $f+1$ vertices are visited and computing the intersection takes time $O(f)$.

The problem is, these trees are huge. Preprocessing them for all pairs of vertices takes total space $O(n^{f+3})$. The main technical contribution of [14] is to reduce the space without sacrificing too much of their performance, that is, the stretch of the reported distance and the query time. In the first step, the number of vertices in the tree is decreased by introducing an approximation parameter $\varepsilon > 0$. Each path P_v is split into $O(\log n/\varepsilon)$ *segments*. Now node v only has a child for each segment and the search procedure recursing on that child corresponds to failing the whole segment instead of only a single edge. This reduces the total size of all trees to $O(n^3\,(c\frac{\log n}{\varepsilon})^f)$ for some constant $c > 0$. However, it leads to some inaccuracies in the answer of the tree. The failed segments may contain edges that

are actually present in $G-F$ and thus the path P_{v^*} stored in the last visited node v^* may take unnecessary detours. It is proven in [14] that $FT(s,t,F) = |P_{v^*}| = d_{G-F}(s,t)$ is correct if all failing edges are "far away"[7] from the true replacement path $P(s,t,F)$ in $G-F$, where the required safety distance depends on the distance $d_{G-F}(s,t)$. To also answer queries for which this condition is violated, they consult multiple FT-trees. An auxiliary graph H^F is constructed on the endpoints $V(F)$ of all failing edges, that is, $V(H^F) = \{s,t\} \cup V(F)$. For each pair of vertices $u, v \in V(H^F)$, the edge $\{u,v\}$ is weighted with the reported distance $FT(u,v,F)$. While not all edge weights may be the correct u-v-replacement distance, the distance of s and t in H^F can be shown to be a $(1+\varepsilon)$-approximation of $d_{G-F}(s,t)$. The idea is that, when going from s to t, one can always find a next vertex in $V(H^F)$ that is not too far off the shortest path and such that the subpath to that vertex is "far away" from all failures. Computing the weights for H^F increases the query time to $O(f^4)$.

The next step is more involved and is concerned with the size of the nodes in the FT-trees. Originally, each of them stores all edges of a path in (a subgraph of) G and therefore may take $O(n)$ space. Afek et al. [1] showed that every shortest path in $G-F$, for $|F| \leqslant f$, is *f-decomposable*, that is, a concatenation of at most f shortest paths in G. Chechik et al. [14] extend this notion to so-called expaths. For a positive integer ℓ, a path is said to be an *ℓ-expath* if it is the concatenation of $(2\log_2(n)+1)$ ℓ-decomposable paths such that the ith ℓ-decomposable path has length at most $\min\{2^i, 2^{2\log_2(n)-i}\}$. Consider a node v in the tree $FT(u,v)$. Instead of storing the shortest u-v-path P_v edge by edge, one would like to represent it by the endpoints of the constituting shortest paths (in G) and edges. However, the collection A_v of edges in all segments that were failed while descending from the root to v may be much larger than f and P_v may not be f-decomposable. Instead, the node v now holds the shortest $(2f+1)$-expath from u to v in $G-A_v$. It can be represented by $O(f\log n)$ endpoints, bringing the total space of the trees to $O(fn^2(\log n)(c\frac{\log n}{\varepsilon})^f)$. It is described in [14] how to navigate the new representation to obtain a $(1+\varepsilon)$-approximation of $d_{G-F}(s,t)$ in time $O(f^5\log n)$.

In this work, we advance the space reduction further into the subquadratic regime. Recall that L is the number of edges up to which a path is called short. When sampling a set B of $\widetilde{O}_\varepsilon(n/L)$ *pivots* uniformly at random, then w.h.p. every long replacement path contains a pivot. Restricting the FT-trees $FT(u,v)$ to only those pairs u, v for which at least one vertex is in B brings the total number of trees to $o(n^2)$. Unfortunately, it deprives us of the replacement distances for pairs that are joined by a short path.

Short Paths. To make up for this deficit, we design an approximate f-DSO for vertex pairs with short replacement paths. We extend a technique by Weimann and Yuster [30] from exact to approximate distances while also reducing the required space and query time. When sampling $\widetilde{O}(L^f)$ spanning subgraphs of G by, in each one, removing any edge independently with probability $1/L$, it is shown in [30] that w.h.p. for each set F of at most f edges and each pair of vertices connected by a short path in $G-F$, there are $\widetilde{O}(1)$ subgraphs that contain the path but none of F. Such a collection of graphs is called an *(L,f)-replacement path covering* (RPC) [25]. For any two vertices s and t that have a replacement path on at most L edges, the minimum s-t-distance of the $\widetilde{O}(1)$ suitable graphs of the RPC is the correct replacement distance.

We cannot use that approach directly in subquadratic space. The subgraphs have total size $\Omega(L^f m)$, which is already too large if G is dense. Also, it is expensive to find the correct members of the RPC for a given query. In [30], the solution was to go over all graphs and explicitly check whether they have the set F removed, dominating the query time (for short paths). Karthik and Parter [25] derandomized this construction and thereby reduced the time needed to find the correct subgraphs to $\widetilde{O}(L)$. Both approaches break down in subquadratic space, since we cannot even store all edges of the graphs. However, we are only seeking *approximate* replacement distances. We exploit this fact in a new way of constructing and approximate (L,f)-replacement path coverings. We do so by turning the sampling technique upside down and combining it with the distance oracle of Thorup and Zwick [29].

Instead of sampling the subgraphs directly by removing edges, we construct them in a hierarchical manner by *adding* connections. We build a tree in which each node is associated with a subset of the edges of G, this set stands for the "missing" edges. We start with the full edge set E in the root, that is, the graph in the root is empty. The height of the tree is h and each node has $L^{f/h}$ children. The associated set of a child node contains any edge of its parent with probability $L^{-1/h}$. This corresponds to adding any missing edge with probability $1-L^{-1/h}$. Knowing the missing edges up front benefits the query algorithm. At each node starting with the root, if we were to expand all children in which all failures of F are missing, we would find the suitable subgraphs. The hierarchical sampling creates some dependencies among the subgraphs associated with the leaves of the tree, while the graphs in [30] were independent. We tackle this issue by always recursing only on one child node and therefore querying a single leaf. We repeat the process in several independent trees in order to amplify the success probability. We prove that there exists a constant $c > 0$ such that $\widetilde{O}(c^h)$ trees together ensure the property we need from an (L,f)-replacement path covering w.h.p. Optimizing the height h gives an $\widetilde{O}(L^{o(1)})$ query time (assuming constant f).

The main challenge is to bring down the size of this construction by reducing the number of edges in the graphs associated with the nodes of the trees. Thorup and Zwick [29] devised, for any positive integer k, a $(2k-1)$-approximate distance oracle together with a compatible spanner of size $O(kn^{1+1/k})$, i.e., the stretched distance returned by the oracle is the length of a shortest path in the spanner. Therefore, we can use the oracles in the leaves of the trees to report distances, giving a low query time, and employ the spanners as proxies for the graphs associated with the intermediate nodes. However, for this to work, we have to carefully tweak the computation of the spanners and interleave it with the sampling process in order to not blow up the size too much (or the stretch).

Long Paths. We return to the fault-tolerant trees. By the use of the pivots, we reduced the required number of trees to $\widetilde{O}_\varepsilon(n^2/L)$. But even in the most compact version of FT-trees, this is not enough to reach subquadratic space all together. The issue is with the representation of expaths as a sequence of $O(f\log n)$ components, each

[7] More formally, a path P being "far away" from F means that, for every vertex x on P except for s and t and every endpoint y of a failing edge in F, the distance from x to y is more than $\frac{\varepsilon}{9}\cdot\min(|P[s,x]|, |P[x,t]|)$, see Definition 5.2.

of which is implicitly represented by its two endpoints. In [14] this was implemented by storing the original graph distance $d(x, y)$ and the predecessor $\text{pred}(x, y)$ of y on the shortest x-y-path for *all* pairs x, y. This information is used to expand the implicit representation of an expath when needed. However, the space is again $\Omega(n^2)$. The key observation to overcome this is that, in our case, we do not need to encode arbitrary expaths but only those with a particular structure, e.g., at least one endpoint is a pivot. This allows us to forgo the need of a quadratic database of all distances.

We also devise a new procedure to obtain an approximation of $d_{G-F}(s, t)$ by combining the values from the FT-trees with the f-DSO for short paths. Recall that we build one FT-tree for each pair of vertices (u, v) where u or v are pivots. The main open issue is to find the weight of the edge $\{u, v\}$ in the auxiliary graph H^F (see above) if neither u nor v are pivots and they also do not have a short path between them in $G - F$. Then, w.h.p. at least one pivot b hits the L-edge prefix of that replacement path. Therefore, it is sufficient to estimate its length as the sum of an approximation for $d^{\leqslant L}_{G-F}(u, b)$ via the f-DSO for short paths, and an approximation for $d_{G-F}(b, v)$ via the FT-trees. However, since we do not know the right pivot b, we have to scan all of them. We prove that this results in a stretch of $3 + \varepsilon$ and a sublinear query time.

While this is already faster than all previous works (for a stretch independent of f), it is still not very efficient. In Section 6, we improve the query time to $O_\varepsilon(n^\alpha)$ for any constant $0 < \alpha < 1/2$. We provide an efficient way to check whether the number of pivots in B that are close to u and v in $G - F$ are below the threshold value of L^{f-1} and, if so, find them all. If only a small number of pivots are around u (or v), we can afford to scan them as described above.

The complementary case of many pivots around both endpoints is solved by precomputing a set of $\widetilde{O}_\varepsilon(n/L^f)$ *new pivots*, much fewer than before, and generalizing the FT-trees to granularity $\lambda > 0$. This ensures that, in any node v, the first and last λ edges of the corresponding path P_v each form their own segment. High granularity thus makes the generalized trees much larger. For comparison, the maximum granularity $\lambda = n$ would unwind *all* of the efforts taken in [14] to reduce their size, as summarized at the beginning of this section. We can still fit the trees in subquadratic space by building $FT_\lambda(b, b')$ only for pairs b, b' of new pivots.

The u-v-distance in $G - F$ in the case of many *original* pivots around u and v is approximated as follows. We compute two *new* pivots b_u, b_v, with b_u close to u in $G - F$ and b_v close to v. The approximate length of the shortest path from u to v in $G - F$ is computed by the overall sum of (i) an approximation of the distance from u to b_u in $G - F$, (ii) an approximation of the distance from b_u to b_v in $G - F$ computed by visiting $FT_\lambda(b_u, b_v)$, and (iii) an approximation of the distance from b_v to v in $G - F$. We make sure to have a granularity $\lambda \leqslant L$ so as to compute the approximations (i) and (iii) using our f-DSO for short paths.

3 PRELIMINARIES

We let $G = (V, E)$ denote the undirected and unweighted base graph with n vertices and m edges. We tacitly assume $m = \Omega(n)$. For any undirected (multi-)graph H, which may differ from the input G, we denote by $V(H)$ and $E(H)$ the set of its vertices and edges, respectively. Let P be a path in H from a vertex $s \in V(H)$ to $t \in V(H)$, we say that P is an *s-t-path* in H. We denote by $|P| = |E(P)|$ the *length* of P. For vertices $u, v \in V(P)$, we let $P[u..v]$ denote the subpath of P from u to v. Let $P = (u_1, \dots, u_i)$ and $Q = (v_1, \dots, v_j)$ be two paths in H. Their *concatenation* is $P \circ Q = (u_1, \dots, u_i, v_1, \dots, v_j)$, which is well-defined if $u_i = v_1$ or $\{u_i, v_1\} \in E(H)$. For $s, t \in V(H)$, the *distance* $d_H(s, t)$ is the minimum length of any s-t-path in H; if s and t are disconnected, we set $d_H(s, t) = +\infty$. When talking about the base graph G, we drop the subscripts.

A *spanning* subgraph of a graph H is one with the same vertex set as H but possibly any subset of its edges. This should not be confused with a spanner. A *spanner of stretch $\sigma \geqslant 1$*, or *σ-spanner*, is a spanning subgraph $S \subseteq H$ such that additionally for any two vertices $s, t \in V(S) = V(H)$, it holds that $d_H(s, t) \leqslant d_S(s, t) \leqslant \sigma \cdot d_H(s, t)$. A *distance oracle* (DO) for H is a data structure that reports, upon query (s, t), the distance $d_H(s, t)$. It has *stretch $\sigma \geqslant 1$*, or is *σ-approximate*, if the reported value $\widehat{d}(s, t)$ satisfies $d_H(s, t) \leqslant \widehat{d}(s, t) \leqslant \sigma \cdot d_H(s, t)$ for any admissible query.

For a set $F \subseteq E$ of edges, let $G-F$ be the graph obtained from G by removing all edges in F. For any two $s, t \in V$, a *replacement path* $P(s, t, F)$ is a shortest path from s to t in $G-F$. Its length $d_{G-F}(s, t)$ is the *replacement distance*. Let L be a positive integer. We call a path in (a subgraph of) G *short* if it has at most L edges, and *long* otherwise. Let $d^{\leqslant L}_{G-F}(s, t)$ be the minimum length of any short s-t-paths in $G - F$, or $+\infty$ if no such path exists.

For a positive integer f, an *f-distance sensitivity oracle* (DSO) answers queries (s, t, F) with $|F| \leqslant f$ with the replacement distance $d_{G-F}(s, t)$. The stretch of a DSO is defined as for DOs. The maximum number f of supported failures is called the *sensitivity*. We measure the space complexity of any data structure in the number of $O(\log n)$-bit machine words. The size of the input graph G does not count against the space, unless it is stored explicitly.

Some proofs and even sections are omitted due to space reasons.

4 HANDLING SHORT PATHS

We develop here our $(2k-1)$-approximate solution for short replacement paths, which will in turn be used for the general distance sensitivity oracle. To do so, we first review (and slightly modify) the distance oracle and spanner in [29] to an extent that is needed to present our construction.

4.1 The Distance Oracle and Spanner of Thorup and Zwick

For any positive integer k, [8] Thorup and Zwick [29] devised a DO that is computable in time $\widetilde{O}(kmn^{1/k})$, has size $O(kn^{1+1/k})$, query time $O(k)$, and a stretch of $2k - 1$. We first review their construction before discussing our changes. First, a family of vertex subsets $V = X_0 \supseteq X_1 \supseteq \cdots \supseteq X_{k-1} \supseteq X_k = \emptyset$ is computed. Each X_i is obtained by sampling the elements of X_{i-1} independently with probability $n^{-1/k}$. We keep this family fixed and apply the construction to a variety of subgraphs of G.

Let H be such a subgraph for which the oracle needs to be computed. For any $v \in V$ and $0 \leqslant i < k$, let $p_{i,H}(v)$ be the closest vertex[9] to v in X_i in the graph H, ties are broken in favor of the

[8] In principle, k could depend on n or m, but for $k = \Omega(\log n)$ we do not get further space improvements. We assume k to be a constant in this work.
[9] We have $p_{i,H}(v) = v$ for all i small enough so that X_i still contains v.

Algorithm 1: Original query algorithm of the distance oracle for pair (s,t).

$w \leftarrow s$;
$i \leftarrow 0$;
while $w \notin \bigcup_{j=0}^{k-1} X_{j,H}(t)$ **do**
  $i \leftarrow i+1$;
  $(s,t) \leftarrow (t,s)$;
  $w \leftarrow p_{i,H}(s)$;
return $d_H(s,w) + d_H(w,t)$;

Algorithm 2: Modified query algorithm of the distance oracle for pair (s,t).

$\widehat{d} \leftarrow \infty$;
for $i = 0$ ***to*** $k-1$ **do**
  if $p_i(s) \in \bigcup_{j=0}^{k-1} X_{i,H}(t)$ **then**
    $\widehat{d} \leftarrow \min\{\widehat{d},\ d_H(s,p_i(s)) + d_H(p_i(s),t)\}$
  if $p_i(t) \in \bigcup_{j=0}^{k-1} X_{i,H}(s)$ **then**
    $\widehat{d} \leftarrow \min\{\widehat{d},\ d_H(t,p_i(t)) + d_H(p_i(t),s)\}$
return $\widehat{d}$;

vertex with smaller label. The distances from v to all elements in $X_{i,H}(v) = \{x \in X_i \mid d_H(v,x) < \min_{y \in X_{i+1}} d_H(v,y)\} \cup \{p_{i,H}(v)\}$ are stored in a hash table. In other words, $X_{i,H}(v)$ contains those vertices of $X_i \setminus X_{i+1}$ that are closer to v then any vertex of X_{i+1}. Note that the set $X_{i,H}(v)$ and vertices $p_{i,H}(v)$ may differ for the various subgraphs of G. This completes the construction of the DO for H.

The oracle is accompanied by a $(2k-1)$-spanner with $O(kn^{1+1/k})$ edges. It stores all those edges of H that lie on a shortest path between v and a vertex in $\bigcup_{0 \leqslant i < k} X_{i,H}(v)$, again ties between shortest paths are broken using the edge labels.

Algorithm 1 shows how the oracle handles the query (s,t). The returned distance can be shown to overestimate $d_H(s,t)$ by at most a factor $2k-1$. We instead use a slightly modified version as presented in Algorithm 2. Observe that the estimate $\widehat{d}$ produced by our version is at most the value returned by the original one and at least the actual distance between s and t. Further, as before, for any s and t, the path corresponding to the new estimate is a concatenation of at most two original shortest paths in H. The interconnecting vertex is either $p_{i,H}(s)$ or $p_{i,H}(t)$ for some i, we denote it as $u_{s,t,H}$, and the $(2k-1)$-approximate shortest path as $P_{s,t,H}$. The reason why we adapt the query algorithm is a crucial inheritance property.

Lemma 4.1 (Inheritance property). *Let $H \subseteq G' \subseteq G$ be two spanning subgraphs of G, $s,t \in V$ two vertices, and $P_{s,t,G'}$ the approximate shortest path underlying the value returned by the (modified) distance oracle for G'. If $P_{s,t,G'}$ also exists in H, then $P_{s,t,H} = P_{s,t,G'}$. Moreover, the oracle for H returns $|P_{s,t,G'}|$ upon query (s,t).*

Proof. Recall that $P_{s,t,G'}$ is a concatenation of two shortest paths in G', say, $P(s,u)$ and $P(u,t)$, where $u = u_{s,t,G'}$ is the interconnecting vertex in $\bigcup_{j<k} X_{j,G'}(s) \cup \bigcup_{j<k} X_{j,G'}(t)$ that minimizes the sum of distances $d_{G'}(s,u) + d_{G'}(u,t)$. Without loosing generality, we have $u = p_{i,G'}(s)$ for some $0 \leqslant i < k$; otherwise, we swap the roles of s and t. Let $0 \leqslant j < k$ be such that $u \in X_{j,G'}(t)$.

For any spanning subgraph $H \subseteq G'$ that contains the path $P_{s,t,G'}$, it holds that $u = p_{i,H}(s)$ and $u \in X_{j,H}(t)$. Here, we use that the tie-breaking for the $p_{i,H}(s)$ does not depend on the edge set of H. Moreover, the shortest s-u-path and u-t-path in the spanner for H are the same as in G, that is, $P(s,u)$ and $P(u,t)$. As a result, we have $u = u_{s,t,H}$ and $P_{s,t,G'} = P_{s,t,H}$. The second assertion of the lemma follows from $d_H(s,u) = |P(s,u)|$ and $d_H(u,t) = |P(u,t)|$. □

4.2 Tree Sampling

We present our fault-tolerant oracle construction for short paths. Recall that a path in G is short if it has at most L edges, and that $d^{\leqslant L}_{G-F}(s,t)$ is the minimum distance over short s-t-paths in $G-F$. Note that, while we assume f and k to be constants, L may depend on m and n. We prove Theorem 1.2 in the remainder of the section.

We first compute the vertex sets $X_0, \dots, X_k$. Define $h = \sqrt{f \ln L}$, $K = \lceil ((2k-1)L)^{f/h} \rceil$, $p = K^{-1/f}$, and $I = C \cdot 11^h \ln n$ for some sufficiently large constant $C > 0$ (independent of f and k). We build I rooted trees $T_1, \dots, T_I$, each of height h, such that any internal node has K children. For the following description, we fix some tree T_i and use x to denote a node in T_i. Let y be the parent of x in case x is not the root. We associate with each x a subset of edges $A_x \subseteq E$ and a spanning subgraph $S_x \subseteq G$ in recursive fashion. For the root of T_i, set $A_x = E$; otherwise A_x is obtained by selecting each edge of A_y independently with probability p. The random choices here and everywhere else are made independently of all other choices.

Let r be the depth of x in T_i (where the root has depth $r = 0$). Define $J_r = 4 \cdot K^{h-r}$ for $r < h$, and $J_h = 1$. The graph S_x is constructed in J_r rounds. In each round, we sample a subset $A \subseteq A_x$ by independently selecting each edge with probability p^{h-r}. We then compute the Thorup-Zwick spanner of $S_y - A$ using the family $X_0, \dots, X_k$. Slightly abusing notation, if x is the root, we define $S_y = G$ here. We set S_x to be the union of all those spanners. Note that, for a leaf x at depth $r = h$, then $A = A_x$ with probability 1, so indeed only $J_h = 1$ iteration is needed.

For each node, we store a dictionary of the edge sets $E(S_x)$ and (except for the root) $A_x \cap E(S_y)$. We use the static construction of Hagerup, Bro Miltersen, and Pagh [24] that, for a set M, has space $O(|M|)$, preprocessing time $\widetilde{O}(|M|)$, and query time $O(1)$. For each leaf of a tree, we store the (modified) distance oracle D_x. At depth $0 \leqslant r \leqslant h$, the tree T_i has K^r nodes. The largest dictionary at depth r is the one for $A_x \cap E(S_y)$ of size $O(J_{r-1} \cdot kn^{1+1/k}) = O(K^{h-r+1} n^{1+1/k})$ (using that k is constant). Due to $K = O((2k-1)^{f/h} L^{f/h})$ and $h = \sqrt{f \ln n}$, we have $K^{h+1} = O(L^{f+o(1)})$ (using that f is constant as well). In total, our data structure requires $O(I \cdot h \cdot K^{h+1} n^{1+1/k}) = \widetilde{O}(L^{f+o(1)} n^{1+1/k})$ space and can be preprocessed in time $O(I \cdot h \cdot K^{h+1}(kmn^{1/k} + kn^{1+1/k})) = \widetilde{O}(L^{f+o(1)} mn^{1/k})$.

4.3 Query Algorithm

Algorithm 3 presents the query algorithm to report approximate distances. Fix a query (s,t,F) where $s,t \in V$ are two vertices and $F \subseteq E$ is a set of at most f edges. For each of the I trees, we start at

Algorithm 3: Algorithm to answer query (s, t, F). D_y is the distance oracle associated with the leaf y.

```
1  d̂ ← ∞;
2  for i = 1 to I do
3      y ← root of T_i;
4      while y is not leaf do
5          foreach child x of y do
6              if F ∩ E(S_y) ⊆ A_x then
7                  y ← x;
8                  continue while-loop;
9          break while-loop;
10     if y is leaf then d̂ ← min{d̂, D_y(s, t)};
11 return d̂;
```

the root and recurse on an arbitrary child, computed in the inner for-loop, that satisfies $F \cap E(S_y) \subseteq A_x$, where y is parent of x. Note that the set A_x is not stored as it may be too large. (We have $|A_x| = m$ in the root.) The test is equivalent to $F \cap E(S_y) \subseteq A_x \cap E(S_y)$ and can be performed in time $O(f)$ using the stored dictionaries. If at some point no child satisfies the condition, the algorithm resumes with the next tree. Once a leaf y is reached, we query the associated (modified) distance oracle D_y with the pair (s, t). Finally, the algorithm returns the minimum of all oracle answers. The query time is $I \cdot O(fhK + k) = \widetilde{O}(L^{o(1)})$.

We are left to prove correctness. That means, we claim that w.h.p. the returned estimate is at least as large as the replacement distance $d(s, t, F)$ and, if s and t are joined by a short path in $G - F$, then this estimate is also at most $(2k-1)\, d^{\leqslant L}_{G-F}(s, t)$. Consider the Thorup-Zwick spanner for $G - F$ and in it the approximate shortest path $P_{s,t,G-F}$ (as defined ahead of Lemma 4.1). If s and t have a short path in $G - F$, then $P_{s,t,G-F}$ has at most $(2k-1)L$ edges.

Let x be a node at depth r in the tree T_i and let S_y be the spanner associated to its parent (or $S_y = G$ if x is the root). We say x is *well behaved* if it satisfies the following three properties.

(1) $F \cap E(S_y) \subseteq A_x$.
(2) Either x is a root or $|E(P_{s,t,G-F}) \cap A_x| < K^{\frac{h-r}{f}}$.
(3) The path $P_{s,t,G-F}$ is contained in S_x.

Our query algorithm follows a path from the root to a leaf node such that at each node Property 1 is satisfied. We show in the following lemma that any child x of a well-behaved node y that fulfills Property 1 is itself well behaved with constant probability.

Lemma 4.2. *The following statements hold for any non-leaf node y in the tree T_i.*

(i) If y satisfies Property 1, then with probability at least $1 - \frac{1}{e}$ there exists a child of y that satisfies Property 1.
(ii) If y satisfies Property 2, then any child of y satisfies Property 2 with probability at least $\frac{1}{4}$.
(iii) If y is well behaved and a child x of y satisfies Properties 1 and 2, then the probability of x being well behaved is at least $1 - \frac{1}{e}$.

The root of T_i is well behaved with probability at least $1 - \frac{1}{e}$.

The next lemma shows that the distance oracle computed for a well-behaved leaf reports a $(2k-1)$-approximation of the distance in $G - F$ for short paths.

Lemma 4.3. *Let $s, t \in V$ be two vertices and $F \subseteq E$ a set of at most f edges. Let further x be a leaf in T_i and D_x be the (modified) distance oracle associated with x. If x satisfies Property 1 with respect to F, then $D_x(s, t) \geqslant d(s, t, F)$. Moreover, if x is well behaved with respect to the approximate shortest path $P_{s,t,G-F}$, then $D_x(s, t) \leqslant (2k-1)\, d(s, t, F)$.*

Proof. As x is a leaf node, S_x is the spanner of the graph $S_y - A_x$ and D_x reports the distances in S_x. By Property 1, we have $F \cap E(S_y) \subseteq A_x$ whence $S_x \subseteq S_y - A_x \subseteq G - F$. This implies that $D_x(s, t) = d_{S_x}(s, t) \geqslant d_{G-F}(s, t) = d(s, t, F)$. If x is even well behaved then, by Property 3, the path $P_{s,t,G-F}$ lies in S_x and thus by inheritance, $D_x(s, t) \leqslant |P_{s,t,G-F}| \leqslant (2k-1) \cdot d(s, t, F)$. □

Our algorithm only ever queries leaves that fulfill Property 1, it therefore never underestimates the distance $d(s, t, F)$. Now additionally assume that s and t are connected in $G - F$ via a path with at most L edges. To complete the proof of Theorem 1.2, we need to show that, under this condition and with high probability over all queries, our algorithm queries at least one well-behaved leaf. If there is a short s-t-path in $G - F$ then $P_{s,t,G-F}$ has at most $(2k-1)L$ edges. Lemma 4.2 shows that the root of each tree T_i, for $1 \leqslant i \leqslant I$, is well behaved with probability $1 - \frac{1}{e}$, and that in each stage the query algorithm finds a well-behaved child node with constant probability. More precisely, we arrive at a well-behaved leaf with probability at least $(1 - \frac{1}{e}) \cdot \left((1 - \frac{1}{e})^2 \frac{1}{4}\right)^h \geqslant \frac{1}{2} \cdot 11^{-h}$. Since there are $I = c \cdot 11^h \ln n$ independent trees, the query algorithms fails for any fixed query with probability at most $(1 - \frac{1}{2 \cdot 11^h})^I \leqslant n^{-c/2}$. We choose the constant $c > 0$ large enough to ensure a high success probability over all $O(n^2 m^f) = O(n^{2+2f})$ possible queries.

5 SUBLINEAR QUERY TIME FOR LONG PATHS

Let $0 < \alpha < 1/2$ be a constant, where the approximation parameter $\varepsilon > 0$ may depend on m and n. As a warm up, we construct a distance sensitivity oracle with the same stretch and space as in Theorem 1.1, but only a sublinear query time of the form $O_\varepsilon(n^{1-g(\alpha,f)})$, for some function g. In Section 6, we then show how to reduce the query time to $\widetilde{O}_\varepsilon(n^\alpha)$. The intermediate solution serves to highlight many of the key ideas needed to implement the classical FT-trees in subquadratic space, but does not yet involve the granularity λ. Recall that we assume that, for every two vertices u and v of G, there is a unique shortest path from u to v in G. Since the short replacement paths are handled by Theorem 1.2, we focus on long paths. The structure of this section is as follows. We first describe the interface of an abstract data structure *FT* and show how to use it to get a $(3+\varepsilon)$-approximation of the replacement distances. We then implement *FT* using FT-trees.

Lemma 5.1. *Let f be a positive integer and $0 < \alpha < 1/2$ a constant. For any undirected, unweighted graph with unique shortest paths and any $\varepsilon > 0$, there exists a $(3+\varepsilon)$-approximate f-DSO that takes space $\widetilde{O}(n^{2-\alpha/(f+1)}) \cdot O(\log n/\varepsilon)^{f+1}$, has query time $n^{1-\frac{\alpha}{f+1}+o(1)}/\varepsilon$, and preprocessing time $\widetilde{O}(n^{2-\alpha/(f+1)}(m + 1/\varepsilon)) \cdot O(\log n/\varepsilon)^f$.*

5.1 Trapezoids and Expaths

For the interface of *FT*, we need a bit of terminology from the work by Chechik et al. [14]. Recall the high-level description of the original FT-trees in Section 2. We now make precise what we mean by all failures in F being "far away" from a given path. Let $0 < \varepsilon < 3$; moreover, it is bounded away from 3 (ε may depend on the input). We use $V(F)$ for the set of endpoints of failing edges.

Definition 5.2 ($\frac{\varepsilon}{9}$-trapezoid). Let $F \subseteq E$ a set of edges, $u, v \in V$, and P a u-v-path in $G - F$. The *$\frac{\varepsilon}{9}$-trapezoid* around P in $G - F$ is

$$\mathrm{tr}^{\varepsilon/9}_{G-F}(P) = \Big\{ z \in V \backslash \{u, v\} \mid \exists y \in V(P) \colon d_{G-F}(y, z) \leqslant \frac{\varepsilon}{9} \cdot \min(\,|P[u..y]|, |P[y..v]|\,) \Big\}.$$

P is *far away*[10] from F if it exists in $G - F$ and $\mathrm{tr}^{\varepsilon/9}_{G-F}(P) \cap V(F) = \emptyset$.

The endpoints u, v of P are removed from the trapezoid to exclude trivialities when applying it to paths between vertices contained in the failing edges. Finally, note that, due to $\varepsilon/9 < 1$, the distance from u to any vertex in the trapezoid is strictly smaller than $d_{G'}(u, v)$ (by symmetry, this also holds for v). The idea is that either the path P is already far away from all failures, or we can reach our destination via a vertex $z \in \mathrm{tr}^{\varepsilon/9}_{G-F}(P) \cap V(F)$ such that the shortest u-z-path in $G - F$ is far away from F and only a slight detour. An illustration is given in **??**.

Lemma 5.3 (Lemma 2.6 in [14]). *Let $u, v \in V(F)$ and $P = P(u, v, F)$ be their replacement path. If $\mathrm{tr}^{\varepsilon/9}_{G-F}(P) \cap V(F) \neq \emptyset$, then there are vertices $x \in \{u, v\}$, $y \in V(P)$, and $z \in \mathrm{tr}^{\varepsilon/9}_{G-F}(P) \cap V(F)$ satisfying*

(i) $|P[x..y]| \leqslant |P|/2$;
(ii) $d_{G-F}(y, z) \leqslant \frac{\varepsilon}{9} \cdot d_{G-F}(x, y)$;
(iii) $\mathrm{tr}^{\varepsilon/9}_{G-F}(P[x..y] \circ P(y, z, F)) \cap V(F) = \emptyset$.

In particular, the path $P[x..y] \circ P(y, z, F)$ is far away from all failures and has length at most $(1 + \frac{\varepsilon}{9}) \cdot d_{G-F}(x, y)$.

We now turn to expaths. Afek et al. [1] showed that shortest paths in $G - F$ are *f-decomposable*, that is, each of them is obtained by concatenating at most $f + 1$ shortest paths in G (for weighted G those shortest paths may be interleaved with up to f edges). One would like to represent replacement paths by the $O(f)$ endpoints of those shortest paths (and edges), but during the construction of the FT-trees much more than f edges may fail, so this is not directly possible. We will see that expath offer a suitable alternative.

Definition 5.4 (ℓ-decomposable path). Let $A \subseteq E$ be a set of edges and ℓ a positive integer. An ℓ-decomposable path in $G - A$ is a path which is the concatenation of at most $\ell + 1$ shortest paths of G.

Definition 5.5 (ℓ-expath). Let $A \subseteq E$ be a set of edges and ℓ a positive integer. An *ℓ-expath* in G–A is a path that is a concatenation of $(2\log_2(n)+1)$ ℓ-decomposable paths such that, for every $0 \leqslant i \leqslant 2\log_2 n$, the length of the i-th path is at most $\min(2^i, 2^{2\log_2(n)-i})$.

Since $n - 1$ is an upper bound on the diameter of any connected subgraph of G, the middle level $i = \log_2 n$ is large enough to accompany any (decomposable) path. Levels may be empty. Therefore, for any $\ell' \geqslant \ell$, an ℓ-decomposable path is also both ℓ'-decomposable and an ℓ'-expath. Also, an arbitrary subpath of an ℓ-decomposable path (respectively, ℓ-expath) is again ℓ-decomposable (respectively, an ℓ-expath). This gives the following intuition why it is good enough to work with expaths. Suppose some replacement path $P(u, v, F)$ survives in G–A albeit $A \supseteq F$ may be much larger than F, then the shortest u-v-path in $G - A$ is indeed $P(u, v, F)$ and thus f-decomposable. The length of the shortest $(2f+1)$-expath between u and v in G–A is the actual replacement distance $|P(u, v, F)| = d_{G-F}(u, v)$. The reason for the choice $\ell = 2f + 1$ will become apparent in the proof of Lemma 5.6. The difficulties of working merely with $(2f+1)$-*decomposable* paths are described in Lemma 5.11.

Finally, we define a set B of special vertices of G that we call *pivots*. Recall that we are mainly interested in paths with more than L edges. Suppose $L = \omega(\log n)$. We construct the set B by sampling any vertex from V independently with probability $C' f \log_2(n)/L$ for some sufficiently large constant $C' > 0$. With high probability, we have $|B| = \widetilde{O}(n/L)$ and any replacement path with more than $L/2$ edges in any of the graphs $G - F$ with $|F| \leqslant f$ contains a pivot as can be seen by standard Chernoff bounds, see e.g. [22, 28, 30].

Interface of Data Structure *FT*. For a positive integer ℓ and vertices $u, v \in V$, define $d^{(\ell)}_{\varepsilon/9}(u, v, F)$ to be the minimum length over all *ℓ-decomposable* paths between u and v in $G - F$ that are far away from F. If there are no such paths, we set $d^{(\ell)}_{\varepsilon/9}(u, v, F) = +\infty$. The data structure *FT* can only be queried with triples (u, v, F) for which u or v is a pivot in B. Its returned value satisfies $d_{G-F}(u, v) \leqslant FT(u, v, F) \leqslant 3 \cdot d^{(2f+1)}_{\varepsilon/9}(u, v, F)$. We let q_{FT} denote its query time.

5.2 Querying the Distance Sensitivity Oracle

We show how to use the black box *FT* to get a $(3+\varepsilon)$-approximate f-DSO. Fix a query (s, t, F) that we want to answer on the top level. Let $u, v \in V$ be any two vertices. Recall that we use $d^{\leqslant L}_{G-F}(u, v)$ for the minimum length over all short u-v-paths in the graph $G - F$, and $\widehat{d^{\leqslant L}}(u, v, F)$ for its $(2k-1)$-approximation by the f-DSO for short paths described in Theorem 1.2. We instantiate that oracle with $k = 2$. The time to obtain the estimate is $\widetilde{O}(L^{o(1)})$.

To answer (s, t, F), we build the complete graph H^F on the vertex set $V(H^F) = \{s, t\} \cup V(F)$ and assign weights to its edges. For a pair $\{u, v\} \in \binom{V(H^F)}{2}$, let $w_{H^F}(u, v)$ denote the weight of the edge $\{u, v\}$. Since G is undirected, $w_{H^F}(\cdot, \cdot)$ is symmetric. We allow possibly infinite edge weights instead of removing the respective edge in order to simplify notation. If u or v is a pivot, we set $w_{H^F}(u, v)$ to the minimum of $\widehat{d^{\leqslant L}}(u, v, F)$ and $FT(u, v, F)$. Otherwise, if $\{u, v\} \cap B = \emptyset$, we set it to the minimum of $\widehat{d^{\leqslant L}}(u, v, F)$ and $w'_{H^F}(u, v) = \min_{b \in B} \{FT(u, b, F) + FT(b, v, F)\}$. The eventual answer to the query (s, t, F) is the distance $d_{H^F}(s, t)$.

Lemma 5.6. *W.h.p. the query time is $\widetilde{O}(L^{o(1)} + \frac{n}{L} \cdot q_{FT})$ and it holds that $d_{G-F}(s, t) \leqslant d_{H^F}(s, t) \leqslant (3+\varepsilon)\, d_{G-F}(s, t)$.*

Proof. The graph H^F has $O(f^2) = O(1)$ edges, and assigning a weight takes $\widetilde{O}(L^{o(1)} + |B| \cdot q_{FT})$ per edge. The distance from s to t can be computed using Dijkstra's algorithm in time $O(f^2)$.

We prove the seemingly stronger assertion that for each pair $u, v \in V(H^F)$, we have $d_{G-F}(u, v) \leqslant d_{H^F}(u, v) \leqslant (3+\varepsilon)\, d_{G-F}(u, v)$.

[10] Our definition relaxes the one in [14] in that we allow $\mathrm{tr}^{\varepsilon/9}_{G-F}(P) \cap \{s, t\} \neq \emptyset$ if $\{s, t\} \not\subseteq V(F)$. This makes the definition independent of the vertices s and t in the query. The proof of Lemma 5.3 remains the same using $V(F)$ instead of $V(H^F)$.

The first inequality is immediate from the fact that the values $\widehat{d^{\leqslant L}}(u,v,F)$, $FT(u,v,F)$, and $FT(u,b,F)+FT(b,v,F)$ for any $b\in B$ are all at least $d_{G-F}(u,v)$.

We prove the second inequality by induction over d_{G-F}. The case $u=v$ is trivial. Assume the inequality holds for all pairs of vertices with replacement distance strictly smaller than $d_{G-F}(u,v)$. We distinguish three cases. In the first case, the (unique) replacement path $P=P(u,v,F)$ has at most L edges. Theorem 1.2 then implies $d_{H^F}(u,v)\leqslant w_{H^F}(u,v)\leqslant \widehat{d^{\leqslant L}}(u,v,F)\leqslant 3\cdot d^{\leqslant L}_{G-F}(u,v)=3\cdot|P|$, which is $3\,d_{G-F}(u,v)$ as P is a replacement path.

If the path P is long instead, it contains a pivot $b\in B$ w.h.p. (possibly $u=b$ or $v=b$). For the second case, assume P has more than L edges and is far away from all failures in F. Note that then the subpaths $P[u..b]$ and $P[b..v]$ are the replacement paths for their respective endpoints, and therefore both f-decomposable (and also $(2f+1)$-decomposable). Moreover, they are far away from all failures as their trapezoids are subsets of $\mathrm{tr}^{\varepsilon/9}_{G-F}(P)$. It holds that

$$\begin{aligned} d_{H^F}(u,v)\leqslant w_{H^F}(u,v)&\leqslant FT(u,b,F)+FT(b,v,F)\\ &\leqslant 3\cdot d^{(2f+1)}_{\varepsilon/9}(u,b,F)+3\cdot d^{(2f+1)}_{\varepsilon/9}(b,v,F)\\ &=3\cdot|P[u..b]|+3\cdot|P[b..v]|=3\cdot d_{G-F}(u,v).\end{aligned}$$

Finally, for the third case suppose the replacement path P is long but *not* far away from F. Lemma 5.3 states the existence of three vertices $x\in\{u,v\}$, $y\in V(P)$, and $z\in\mathrm{tr}^{\varepsilon/9}_{G-F}(P)\cap V(F)$ such that $d_{G-F}(z,y)\leqslant\frac{\varepsilon}{9}\cdot d_{G-F}(x,y)$. The path $P'=P[x..y]\circ P(y,z,F)$ is far away from all failures and has length at most $(1+\frac{\varepsilon}{9})\cdot d_{G-F}(x,y)$. In the remainder, we assume $x=u$; the argument for $x=v$ is symmetric. If the concatenation P' has at most L edges, we get $w_{H^F}(u,z)\leqslant\widehat{d^{\leqslant L}}(u,z,F)\leqslant 3|P'|\leqslant 3\left(1+\frac{\varepsilon}{9}\right)d_{G-F}(u,y)$.[11] The latter is equal to $(3+\frac{\varepsilon}{3})\,d_{G-F}(u,y)$.

Note that $P[u..y]$ is in fact the unique replacement path $P(u,y,F)$. So, if P' has more than L edges, one of its subpaths $P[u..y]$ or $P(y,z,F)$ has more than $L/2$ edges. Thus, there exists a pivot b on P'. Here, we actually use the uniqueness of shortest paths in G since replacing, say, $P[u..y]$ with another shortest u-y-path in $G-F$ to ensure a pivot may result in a concatenation that is no longer far away from all failures. Similar to the second case, we arrive at

$$\begin{aligned} w_{H^F}(u,z)&\leqslant FT(u,b,F)+FT(b,z,F)\\ &\leqslant 3\cdot d^{(2f+1)}_{\varepsilon/9}(u,b,F)+3\cdot d^{(2f+1)}_{\varepsilon/9}(b,z,F)\\ &\leqslant 3\cdot|P'[u..b]|+3\cdot|P'[b..z]|\leqslant\left(3+\frac{\varepsilon}{3}\right)d_{G-F}(u,y).\end{aligned}$$

It is important that FT approximates $d^{(2f+1)}_{\varepsilon/9}$ because P' may not be f-decomposable, but it is the concatenation of two f-decomposable paths and thus $(2f+1)$-decomposable; so are $P'[u..b]$ and $P'[b..z]$.

Now that we have an upper bound on $w_{H^F}(u,z)$ we can conclude the third case. Since $\frac{\varepsilon}{9}<1$ and $z\in\mathrm{tr}^{\varepsilon/9}_{G_F}(P)$ (where P is the u-v-replacement path), the distance $d_{G-F}(z,v)$ is strictly smaller than $d_{G-F}(u,v)$. By induction, $d_{H^F}(z,v)\leqslant(3+\varepsilon)\cdot d_{G-F}(z,v)$. Recall that vertex y lies on P, whence $d_{G-F}(u,y)+d_{G-F}(y,v)=d_{G-F}(u,v)$. Due to $\varepsilon\leqslant 3$, we have $(2+\frac{\varepsilon}{3})\frac{\varepsilon}{9}\leqslant\frac{\varepsilon}{3}$. Also, recall that $d_{G-F}(z,y)\leqslant\frac{\varepsilon}{9}d_{G-F}(u,y)$ by the definition of z and $x=u$. Putting everything together, we estimate the u-v-distance in the graph H^F.

$$\begin{aligned} &d_{H^F}(u,v)\leqslant w_{H^F}(u,z)+d_{H^F}(z,v)\\ &\leqslant\left(3+\frac{\varepsilon}{3}\right)d_{G-F}(u,y)+(3+\varepsilon)d_{G-F}(z,v)\\ &=3\left(\left(1+\frac{\varepsilon}{9}\right)d_{G-F}(u,y)+\left(1+\frac{\varepsilon}{3}\right)(d_{G-F}(z,y)+d_{G-F}(y,v))\right)\\ &\leqslant 3\left(\left(1+\frac{\varepsilon}{9}\right)d_{G-F}(u,y)+\left(1+\frac{\varepsilon}{3}\right)\left(\frac{\varepsilon}{9}d_{G-F}(u,y)+d_{G-F}(y,v)\right)\right)\\ &=3\left(d_{G-F}(u,y)+d_{G-F}(y,v)+\left(2+\frac{\varepsilon}{3}\right)\frac{\varepsilon}{9}d_{G-F}(u,y)+\frac{\varepsilon}{3}d_{G-F}(y,v)\right)\\ &\leqslant 3\left(d_{G-F}(u,v)+\frac{\varepsilon}{3}d_{G-F}(u,y)+\frac{\varepsilon}{3}d_{G-F}(y,v)\right).\end{aligned}$$

The last expression is equal to $(3+\varepsilon)d_{G-F}(u,v)$. □

5.3 Fault-Tolerant Trees

We now describe the implementation of the FT data structure via fault-tolerant trees. We compute all-pairs shortest distances in the original graph G (if required, with perturbed edge weights for unique shortest paths), and, for each pivot $b\in B$, a shortest path tree of G rooted in b in $\widetilde{O}(mn)$ time. We turn each of those trees into an data structure that reports the lowest common ancestor (LCA) in constant time with the algorithm of Bender and Farach-Colton [4]. This takes time and space $O(|B|n)=\widetilde{O}(n^2/L)$ w.h.p.

We also assume that we have access to a procedure that, given any set $A\subseteq E$ of edges (which may have much more than f elements) and pair of vertices $u,v\in V$, computes the shortest $(2f+1)$-expath between u and v in $G-A$. This expath is labeled with its structure, that means, (a) the start and endpoints of the $2\log_2(n)+1$ constituting $(2f+1)$-decomposable subpaths, and (b) inside each decomposable path the start and endpoint of the constituting shortest paths (and possibly interleaving edges). We explain in the full version how to how to achieve this in time $\widetilde{O}(fm)$, this is also the key ingredient of the proof of Theorem 1.4.

We build the FT-trees only for pairs of vertices (u,b) for which $b\in B$ is a pivot. On a high level, $FT(u,b)$ is a tree of depth f that stores in each node the shortest $(2f+1)$-expath between u and b in some graph $G-A$. We first describe the information that we hold in a single node v. Let P_v be the stored expath. It is partitioned first into segments and those into parts. To define the segments, we need the notion of netpoints.

Definition 5.7 (Path netpoints). Let $P=(u=v_1,\dots,v_\ell=b)$ be a path. Define p_{left} to be all vertices $v_j,v_{j+1}\in V(P)$ such that $|P[u..v_j]|<(1+\frac{\varepsilon}{36})^i\leqslant|P[u..v_{j+1}]|$ for some integer $i\geqslant 0$. Analogously, let p_{right} be all vertices $v_j,v_{j-1}\in V(P)$ such that $|P[v_j..b]|<(1+\frac{\varepsilon}{36})^i\leqslant|P[v_{j-1}..b]|$ for some i. The *netpoints* of P are all vertices in $p_{\text{left}}\cup p_{\text{right}}\cup\{u,b\}$.

A *segment* of the path P is the subpath between consecutive netpoints. For an edge $e\in E(P)$, let $\mathrm{seg}(e,P)$ denote the segment of P containing e. The netpoints cut P into segments of exponentially increasing length, with $1+\frac{\varepsilon}{36}$ being the base of the exponential. However, since we do this from *both* ends the segments do not get too large. We make this precise in Lemma 5.10 below.

The segments are further subdivided into parts. An expath P consists of decomposable subpaths, which in turn consist of shortest

[11] We do mean $d_{G-F}(u,y)$ here and *not* $d_{G-F}(u,z)$.

paths (and interleaving edges) in G, but they may not be aligned with the segments. To avoid this, we define a *part* of P to be a maximal subpath that is completely contained in a shortest path of a $(2f+1)$-decomposable subpath and also does not cross netpoints. We can find all parts by a linear scan over the labels of the expath given by the procedure mentioned above. Note that each part is a shortest path/edge in G. By the assumption that shortest paths are unique, it is enough to represent a part by its two endpoints. With each part $[v, w]$, for $v, w \in V(P)$, we keep the information whether one or both endpoints are netpoints, store the original graph distance $d(v, w)$, and a marker whether that part is long, i.e., whether it contains more than L edges. If so, we additionally store a pivot $p \in B$ that lies in that part. (The case $p = b$ is possible if the respective part lies at the end of the expath P, that is, if $w = b$.)

We now describe the whole FT-tree $FT(u, b)$ recursively. In some node ν, let A_ν be the set of all edges that were failed in the path from the root to ν; with $A_\nu = \emptyset$ in the root itself. We compute the shortest $(2f+1)$-expath P_ν in $G - A_\nu$ and store the information for all its parts. For each of its segments S, we create a child node μ in which we set $A_\mu = A_\nu \cup E(S)$. That means, the transition from a parent to a child corresponds to failing the *whole segment*. Note that the sets A_ν are only used during preprocessing and never actually stored. We continue the recursive construction until depth f is reached; if in a node ν the vertices u and b become disconnected, we mark this as a leaf node not storing any path. We build one FT-tree for each pair of (distinct) vertices in $V \times B$ and additionally store the LCA data structure for each pivot.

The number of segments of any simple path in a subgraph of G is at most $2\log_{1+\frac{\varepsilon}{36}}(n) + 1$. Therefore, there exists a constant $c > 0$ such that the maximum number of segments of one path is at most $c \log_2(n)/\varepsilon$. This is an upper bound on the degree of any node, so there are at most $(c \log_2(n)/\varepsilon)^f$ nodes in each tree. Moreover, an $(2f+1)$-expath consists of $O(f \log n)$ shortest paths. So there are $O(f \log^2 n/\varepsilon)$ parts in one node, for each of which we store a constant number of machine words. In summary, all FT-trees and LCA structures together take space

$$|B|n \cdot O\left(\frac{f \log^2 n}{\varepsilon}\right) \cdot O\left(\frac{\log n}{\varepsilon}\right)^f + O(|B|n) = \widetilde{O}\left(\frac{n^2}{L}\right) \cdot O\left(\frac{\log n}{\varepsilon}\right)^{f+1}.$$

The time spent in each node is dominated by computing the $(2f+1)$-expath. The total time to precompute FT is $|B|n \cdot \widetilde{O}\left(fm + \frac{1}{\varepsilon}\right) \cdot O\left(\frac{\log n}{\varepsilon}\right)^f + O(|B|n) = \widetilde{O}\left(\frac{n^2}{L}\left(m + \frac{1}{\varepsilon}\right)\right) \cdot O\left(\frac{\log n}{\varepsilon}\right)^f$.

5.4 Querying the Data Structure *FT*

We used in Lemma 5.6 that the returned value $FT(u, b, F)$ is at least $d_{G-F}(u, b)$ and most $3\, d^{(2f+1)}_{\varepsilon/9}(u, b, F)$, three times the minimum length of an $(2f+1)$-decomposable between u and b in $G-F$ that is far away from all failures in F. We now show how to do this.

The main challenge when traversing the FT-tree is to utilize the little information that is stored in a node ν to solve the following problem. Either find the segment $\mathrm{seg}(e, P_\nu)$ for some failing edge $e \in F$ or verify that $F \cap E(P_\nu) = \emptyset$. The original solution in [14] was to compare for each shortest path/interleaving edge $[v, w]$ on P_ν and edge $e = \{x, y\} \in F$ whether the minimum of $d(v, x) + w(x, y) + d(y, w)$ and $d(v, y) + w(x, y) + d(x, w)$ is equal to $d(v, w)$. If so, e must lie on the shortest path $P_\nu[v..w]$. Finding the according segment amounts to computing the two bounding netpoints with a binary search. The problem is that this approach requires to store all $\Omega(n^2)$ original graph distances in G, which we cannot afford. We first prove that we can get a weaker guarantee with our setup.

Lemma 5.8. *Let ν be a node of $FT(u, b)$. There exists an algorithm to check that there is a path between u and b in $G - F$ that has length at most $3|P_\nu|$ or find the segment $\mathrm{seg}(e, P_\nu)$ for some $e \in F \cap E(P_\nu)$. The computation time is $\widetilde{O}(L^{o(1)}/\varepsilon)$.*

Proof. Note that one of the alternatives must occur for if $F \cap E(P_\nu) = \emptyset$, then P_ν exists in $G - F$. Consider a part $[v, w]$ of P_ν. If it has more than L edges, then we stored a pivot p in $[v, w]$. More precisely, $[v, w]$ is the concatenation of the unique shortest path between v and p and the one between p and w in G. We have access to a shortest path tree rooted in p. So, for each edge $e = \{x, y\} \in F$, we can check with a constant number of LCA queries involving p, v, w, x, and y whether edge e is in that concatenation in time $O(f)$ per part. If all checks fail, we have $d_{G-F}(v, w) = d(v, w) = |P_\nu[v..w]|$.

If $[v, w]$ is short, the oracle from Theorem 1.2 is queried with the triple (v, w, F). That oracle was preprocessed anyway and answers in time $\widetilde{O}(L^{o(1)})$. The return value $\widehat{d^{\leqslant L}}(v, w, F)$ is compared with the original distance $d(v, w)$ that was stored with the part. If the former is more than 3 times the latter, it must be that $d_{G-F}(v, w) > d(v, w)$, so the part contains some edge of F.

We either find a part that has a failing edge in total time $\widetilde{O}(L^{o(1)} \cdot f\frac{\log^2 n}{\varepsilon}) = \widetilde{O}(L^{o(1)}/\varepsilon)$ or verify that $d_{G-F}(v, w) \leqslant 3 \cdot d(v, w)$ holds for *all* parts. In the latter case, swapping each part $[v, w]$ by its replacement path $P(v, w, F)$ shows the existence of a path in $G - F$ of length at most $3|P_\nu| = \sum_{[v,w]} 3d(v, w)$. Finally, let $[v, w]$ be a part with $E([v, w]) \cap F \neq \emptyset$. It is completely contained in a segment, finding the two closest netpoints on the subpaths $P_\nu[u..v]$ and $P_\nu[w..b]$ takes only $O(\log n/\varepsilon) = \widetilde{O}(1/\varepsilon)$ additional time. □

We use the lemma to compute $FT(u, b, F)$. The tree transversal starts at the root. Once it enters a node ν, it checks whether there is a path in $G - F$ of length at most $3|P_\nu|$. If so, this length is returned. Otherwise, the algorithm obtains a segment $\mathrm{seg}(e, P_\nu)$ for some $e \in F \cap E(P_\nu)$ and recurses on the corresponding child. Once a leaf ν^* is encountered, the length $|P_{\nu^*}|$ is returned; or $+\infty$ if the leaf does not store a path. This takes total time $q_{FT} = \widetilde{O}(L^{o(1)}/\varepsilon)$ since at most $f+1 = O(1)$ nodes are visited. The main argument for the correctness of this procedure is to show that if a $(2f+1)$-expath P in $G - F$ is far away from all failures, it survives in $G - A_{\nu^*}$.

Lemma 5.9. *Let P be the shortest $(2f+1)$-decomposable path between u and b in $G - F$ that is far away from all failures in F. Let ν^* be the node of $FT(u, b)$ in which a value is returned when queried with F, and let A_{ν^*} be the set of edges that were failed from the root to ν^*. Then, P exists in the graph $G - A_{\nu^*}$. Moreover, it holds that $d_{G-F}(u, b) \leqslant FT(u, b, F) \leqslant 3 \cdot d^{(2f+1)}_{\varepsilon/9}(u, b, F)$.*

We need the following two lemmas for the proof. The first one states that the segments of a path are not too long, or even only contain a single edge. The second lemma verifies a certain prefix optimality of expaths. This is the crucial property that decomposable paths are lacking. For some edge set $A \subseteq E$, let $d^{(\ell)}(u, v, A)$ be the

length of the shortest ℓ-*decomposable* path in $G-A$. Compared to $d^{(\ell)}_{\varepsilon/9}(u,v,F)$, this definition allows for larger failure sets and drops the requirement of the path being far away from the failures.

Lemma 5.10 (Lemma 3.2 in [14]). *Let $u \in V$ and $b \in B$, P be any path between u and b, $e \in E(P)$, and y a vertex of the edge e. Then, $E(\mathrm{seg}(e,P)) = \{e\}$ or $|\mathrm{seg}(e,P)| \leqslant \frac{\varepsilon}{36}\min(|P[u..y]|, |P[y..b]|)$.*

Lemma 5.11 (Lemma 3.1 in [14]). *Let $u \in V$ and $b \in B$, $A \subseteq E$ a set of edges, ℓ a positive integer, and P the shortest ℓ-expath between u and b in $G-A$. Then, for every $y \in V(P)$, $|P[u..y]| \leqslant 4 \cdot d^{(\ell)}(u,y,A)$ and $|P[y..v]| \leqslant 4 \cdot d^{(\ell)}(y,v,A)$ both hold.*

Proof of Lemma 5.9. The second assertion is an easy consequence of the first. P is the shortest $(2f+1)$-decomposable u-b-path in $G-F$ that is far away from all failures in F. If P also exists in $G-A_{v^*}$, then $|P_{v^*}| \leqslant |P|$ by the definition of P_{v^*} as the shortest $(2f+1)$-expath between u and v in $G-A_{v^*}$ and P being $(2f+1)$-decomposable (and thus a $(2f+1)$-expath). The query algorithm guarantees $FT(u,b,F) \leqslant 3|P_{v^*}| \leqslant 3|P| = 3 \cdot d^{(2f+1)}_{\varepsilon/9}(u,b,F)$. It is clear that we never underestimate the true distance $d_{G-F}(u,b)$.

We show the existence of the path P in $G-A_v$ for *every* visited node v by induction over the parent-child transitions of the tree transversal. It is true for the root where $A_v = \emptyset$. When going from v to a child, A_v gets increased by the edges $E(\mathrm{seg}(e,P_v))$ of a segment for some $e_F \in F \cap E(P_v)$. It is enough to prove that P does not contain an edge of $\mathrm{seg}(e,P_v)$. Intuitively, we argue that the segments are too short for their removal to influence a path far away from F.

The claim is immediate if $E(\mathrm{seg}(e_F,P_v)) = \{e_F\}$, because P exists in $G-F$. For the remainder, suppose $\mathrm{seg}(e_F,P_v)$ consists of more than one edge. To reach a contradiction, assume $e_P \in E(P) \cap E(\mathrm{seg}(e_F,P_v))$ is an edge in the intersection. If $\mathrm{seg}(e_F,P_v)$ contains multiple edges from F, we let e_F be the one closest to e_P. This ensures that the subpath of P_v between the closest vertices in e_F and e_P does not contain any other failing edges. More formally, there are vertices $y \in e_P$ and $z \in e_F$ such that neither y nor z are the endpoints u or b and the subpath $P_v[y..z]$ lies entirely both in $\mathrm{seg}(e,P_v)$ and the graph $G-F$. Since $y \in V(P)$, $z \in V(F)$, and the path P is far away from all failures, z must be outside of the trapezoid $\mathrm{tr}^{\varepsilon/9}_{G-F}(P)$, that is, $|\mathrm{seg}(e_F,P_v)| \geqslant |P_v[y..z]| \geqslant d_{G-F}(y,z) > \frac{\varepsilon}{9}\min(|P[u..y]|, |P[y..b]|)$. Conversely, we combine Lemmas 5.10 and 5.11, the fact that edge e_P lies both on P and P_v, as well as P_v (with its subpaths) being a $(2f+1)$-expath to arrive at

$$\begin{aligned}|\mathrm{seg}(e_F,P_v)| &\leqslant \frac{\varepsilon}{36}\min(|P_v[u..y]|, |P_v[y..b]|)\\ &\leqslant \frac{\varepsilon}{36}\cdot\min\Big(4\cdot d^{(2f+1)}(u,y,F),\ 4\cdot d^{(2f+1)}(y,b,F)\Big)\\ &\leqslant \frac{\varepsilon}{9}\cdot\min(|P[u..y]|, |P[y..b]|).\end{aligned}$$

□

5.5 Proof of Lemma 5.1

We derive the parameters of the f-DSO with sublinear query time. The preprocessing consists of two main parts. First, the oracle for short paths is computable in time $\widetilde{O}(L^{f+o(1)}\, m\sqrt{n})$ (Theorem 1.2). Secondly, FT has preprocessing time $\widetilde{O}((n^2/L)(m+1/\varepsilon))O(\log n/\varepsilon)^f$, assuming that we can compute expaths in time $\widetilde{O}(fm)$. We set $L = n^{\alpha/(f+1)}$ for a constant $0 < \alpha < 1/2$. The total preprocessing time is dominated by the FT-trees giving a total time of $O(n^{2-\frac{\alpha}{f+1}}(m+1/\varepsilon))\cdot O(\log n/\varepsilon)^f$. By Lemma 5.6 with $q_{FT} = \widetilde{O}(L^{o(1)}/\varepsilon)$ the query time of the resulting oracle is $\widetilde{O}(n/\varepsilon L^{1-o(1)}) = n^{1-\frac{\alpha}{f+1}+o(1)}/\varepsilon$. The data structure from Theorem 1.2 requires space $\widetilde{O}(L^{f+o(1)}n^{3/2})$, and FT takes $\widetilde{O}(n^2/L)\cdot O(\log n/\varepsilon)^{f+1}$. Inserting our choice of L gives $n^{\frac{f}{f+1}\alpha+\frac{3}{2}+o(1)} + \widetilde{O}\Big(n^{2-\frac{\alpha}{f+1}}\Big)\cdot O\Big(\frac{\log n}{\varepsilon}\Big)^{f+1}$. Since $\alpha < 1/2$ is a constant, the second term dominates.

6 REDUCING THE QUERY TIME

We now reduce the query time to $O_\varepsilon(n^\alpha)$. The bottleneck of the query answering is computing the (auxiliary) weight $w'_{H^F}(u,v)$ of the edge $\{u,v\}$ in the graph H^F, see the beginning of Section 5.2. Minimizing $FT(u,b,F) + FT(b,v,F)$ over all pivots b takes linear time in $|B|$. Let $\lambda = \lambda(L,\varepsilon) \leqslant L$ be a parameter to be fixed later. We define $\mathrm{ball}_{G-F}(x,\lambda) = \{z \in V \mid d_{G-F}(x,u) \leqslant \lambda\}$. If we had access to the graph $G-F$ at query time, we could run Dijkstra's algorithm from u and from v to scan the balls $\mathrm{ball}_{G-F}(u,\lambda)$ and $\mathrm{ball}_{G-F}(v,\lambda)$ of radius λ, and only consider the pivots that are inside these balls. By carefully adapting the sampling probability of the pivots to $\widetilde{O}(n/\lambda)$, we ensure at least one of them hits the shortest expath (replacement path) for from u to v w.h.p. (more details below). The problem is that these balls may still contain too many pivots. In the worst case, we have, say, $\mathrm{ball}_{G-F}(u,\lambda) \cap B = B$ degenerating again to the need to scan *all* pivots. Furthermore, we cannot even afford storing all balls as there are $\Omega(nm^f)$ different ones, a ball for each pair (x,F). Finally, the assumption of access to $G-F$ itself is problematic in the subquadratic-space regime.

To handle all these issues, we consider two cases when computing $w'_{H^F}(u,v)$. That of *sparse balls*, where at least one of $\mathrm{ball}_{G-F}(u,\lambda)$ and $\mathrm{ball}_{G-F}(v,\lambda)$ contains less that L^f vertices, and the case of *dense balls* where the two sets both contain more than L^f vertices.

6.1 The Case of Sparse Balls

Consider the same setup as in Section 5.1, only that the pivots for B are now sampled with probability $C''f\log_2(n)/\lambda$ for some $C'' > 0$. By making the constant C'' slightly larger than C' in the original sampling probability (see the end of Section 5.1), we ensure that w.h.p. every path that is a *concatenation* of at most two replacement paths and has more than λ edges contains a pivot. (Previously, we only had this for ordinary replacement paths with at least $L/2$ edges.) Note that all statements from Section 5 except for the space, preprocessing and query time in Lemma 5.1 remain true. Further, observe that in the case of sparse balls, w.h.p. there are $\widetilde{O}(L^f/\lambda)$ pivots in $\mathrm{ball}_{G-F}(u,\lambda)$ or in $\mathrm{ball}_{G-F}(v,\lambda)$. In this case, it is sufficient to scan those in the same way as we did above. The only issue is that we do not have access to $\mathrm{ball}_{G-F}(u,\lambda)$ at query time, so we precompute a proxy.

Let $G_1,\ldots,G_\kappa$ be all the subgraphs of G in the leaves of the sampling trees introduced in Section 4.2. Recall that they form an (L,f)-replacement path covering w.h.p. During preprocessing, we compute and store the sets $B_{G_i}(x,\lambda) = B \cap \mathrm{ball}_{G_i}(x,\lambda)$ for all the sparse balls $\mathrm{ball}_{G_i}(x,\lambda)$, that is, if $|\mathrm{ball}_{G_i}(x,\lambda)| \leqslant L^f$. Otherwise,

we store a marker that $\text{ball}_{G_i}(x,\lambda)$ is dense.[12] As $\kappa = L^{f+o(1)}$ and w.h.p. $|B_{G_i}(x,\lambda)| = \widetilde{O}(L^f/\lambda)$ for sparse balls, storing all of these sets requires $\widetilde{O}(nL^{2f+o(1)}/\lambda)$ space. One can compute $B_{G_i}(x,\lambda)$ by running Dijkstra from x in G_i until at most L^f vertices are discovered in time $\widetilde{O}(L^{2f})$. In total, this takes $\widetilde{O}(nL^{3f+o(1)})$ time.

Suppose we want to compute the weight $w_{H^F}(u,v)$ in the sparse balls case, meaning that there is an $x \in \{u,v\}$ such that the true set $\text{ball}_{G-F}(x,\lambda)$ is sparse.[13] We use y to denote the remaining vertex in $\{u,v\}\backslash\{x\}$. Let $i_1,\dots,i_r$ be the indices of the graphs G_{i_j} that exclude F as computed by Algorithm 3. We showed in Section 4.2 that $r = \widetilde{O}(L^{o(1)})$ and that the indices can be found in time proportional to their number. By definition of x, all the proxies $\text{ball}_{G_{i_j}}(x,\lambda)$ for $1 \leqslant j \leqslant r$ are sparse as well. Departing from Section 5.2, we set

$$w'_{H^F}(u,v) = \min_{\substack{1\leqslant j\leqslant r \\ b\in B_{G_{i_j}}(x,\lambda)}} \left(\widehat{d^{\leqslant L}}(x,b,F) + FT(b,y,F)\right). \quad (1)$$

The (actual) weight is $w_{H^F}(u,v) = \min(w'_{H^F}(u,v), \widehat{d^{\leqslant L}}(u,v,F))$. Its computation takes time $\widetilde{O}(L^{f+o(1)}/\varepsilon\lambda)$ as there are $\widetilde{O}(L^{o(1)})$ balls, each with $\widetilde{O}(L^f/\lambda)$ pivots, the values $\widehat{d^{\leqslant L}}$ can be evaluated in time $L^{o(1)}$ (Theorem 1.2), and we navigate through $\widetilde{O}(L^f/\lambda)$ FT-trees with a query time of $q_{FT} = \widetilde{O}(L^{o(1)}/\varepsilon)$ each.

Recall the proof of the $(3+\varepsilon)$-approximation from Lemma 5.6. Clearly, if the replacement path $P(u,v,F)$ is short, then $d_{H^F}(u,v) \leqslant 3 \cdot d_{G-F}(u,v)$ still holds since the argument is independent of the definition of $w'_{H^F}(u,v)$. We make the next step in recovering what was dubbed the "second case" for sparse balls. (The proof needs the transition from $\widetilde{O}(n/L)$ to $\widetilde{O}(n/\lambda)$ pivots.)

Lemma 6.1. *Let $u,v \in V$ be such that $|\text{ball}_{G-F}(u,\lambda)| \leqslant L^f$ or $|\text{ball}_{G-F}(v,\lambda)| \leqslant L^f$, and the replacement path $P(u,v,F)$ is long and far away from all failures in F. Then, with high probability $w_{H^F}(u,v) \leqslant 3 \cdot d_{G-F}(u,v)$ holds.*

6.2 The Case of Dense Balls

To transfer the proof of Lemma 5.6, $w_{H^F}(u,v) \leqslant 3\, d_{G-F}(u,v)$ would have to be true also if both $\text{ball}_{G-F}(u,\lambda)$ and $\text{ball}_{G-F}(v,\lambda)$ are dense and $P(u,v,F)$ is far away from all failures. If that were our only concern, Equation (1), would ensure that. However, the query time is $\Omega(n/\lambda)$ since a dense ball may contain too many pivots. We provide a more efficient query algorithm which, however, only gives a $(3+\delta)$-approximation for a small $\delta > 0$ (Lemma 6.6). Therefore, we have to adapt the proof of Lemma 5.6.

Our changes to the construction are twofold. We define a set $\mathcal{B}$ of *new* pivots, polynomially sparser than B, by sampling each vertex independently with probability $C' f \log_2 n/\lambda L^{f-1}$. By a Chernoff bound and $\lambda L^{f-1} \leqslant L^f$, it holds that w.h.p. $|\mathcal{B}| = \widetilde{O}(n/\lambda L^{f-1})$ and all sets $\text{ball}_{G-F}(x,\lambda)$ with $|\text{ball}_{G-F}(x,\lambda)| > L^f$ contain a new pivot. We build an FT-tree with granularity λ for each pair in $\mathcal{B}^2$.

FT-Trees with Granularity. Given two new pivots $b_u, b_v \in \mathcal{B}$, we denote by $FT_\lambda(b_u,b_v)$ the *fault-tolerant tree of b_u and b_v with granularity λ*. Granularity affects the netpoints, segments and expaths.

Definition 6.2 (Path netpoints with granularity λ). Let $P = (b_u = v_1, v_2, \dots, v_\ell = b_v)$ be a path. If $|P| \leqslant 2\lambda$, then the *netpoints of P with granularity λ* are all vertices $V(P)$ of the path. Otherwise, define p_{left} to be all vertices $v_j, v_{j+1} \in V(P)$ with $\lambda \leqslant j \leqslant \ell - \lambda$ such that $|P[v_\lambda..v_j]| < (1+\frac{\varepsilon}{36})^i \leqslant |P[v_\lambda..v_{j+1}]|$ for some integer $i \geqslant 0$. Analogously, let p_{right} be all vertices $v_j, v_{j-1} \in V(P)$ such that $|P[v_j..v_{\ell-\lambda}]| < (1+\frac{\varepsilon}{36})^i \leqslant |P[v_{j-1}..v_{\ell-\lambda}]|$ for some i. The *netpoints of P with granularity λ* are all vertices in $\{v_0,\dots,v_\lambda\} \cup p_{\text{left}} \cup p_{\text{right}} \cup \{v_{\ell-\lambda},\dots,v_\ell\}$.

For $\lambda = 0$, this is the same as Definition 5.7. Similar as before, we denote by $\text{seg}_\lambda(e,P)$ for $e \in P$ the set of *segment* w.r.t. to the new netpoints that contains e. Any path has $O(\lambda)+O(\log_{1+\varepsilon} n) = O(\lambda)+O(\log n/\varepsilon)$ netpoints with granularity λ and thus so many segments. The number of nodes per tree is now $(O(\lambda)+O(\log n/\varepsilon))^f = O(\lambda^f)+O(\log n/\varepsilon)^f$ In summary, the crucial change is that the first and last λ edges are in their own segment and the segment lengths increase exponentially only in the middle part.

Definition 6.3 (ℓ-expath with granularity λ). Let $A \subseteq E$ be a set of edges and ℓ a positive integer. An *ℓ-expath with granularity λ* in $G-A$ is a path $P_a \circ P_b \circ P_c$ such that P_a and P_c contain at most λ edges each, while P_b is a concatenation of $(2\log_2(n)+1)$ ℓ-decomposable paths such that, for every $0 \leqslant i \leqslant 2\log_2 n$, the length of the i-th ℓ-decomposable path is at most $\min(2^i, 2^{2\log_2(n)-i})$.

The *parts* of an ℓ-expath with granularity λ are defined as before. In each node v of $FT_\lambda(b_u,b_v)$, we store the shortest $(2f+1)$-expath P_v with granularity λ from b_u to b_v in G_v. Note that P_v now has $O(f\log(n)\cdot(\lambda+\log(n)/\varepsilon))$ many parts.

Space and Preprocessing Time. Recall the analysis at the end of Section 5.3, and also that we changed the size of $|B|$ to $\widetilde{O}(n/\lambda)$. The number of nodes in $FT_\lambda(b_u,b_v)$ is $O(\lambda^f)+O(\log n/\varepsilon)^f$ and we only need $|\mathcal{B}|^2 = \widetilde{O}(n^2/\lambda^2L^{2f-2})$ new trees. With $\lambda \leqslant L$, this makes for $\widetilde{O}\left(\frac{n^2}{L^f}\right) + \widetilde{O}\left(\frac{n^2}{\lambda^2L^{2f-2}}\right) O\left(\frac{\log n}{\varepsilon}\right)^f$ nodes in all new trees. This is less than the $\widetilde{O}(n^2/\lambda)\cdot O(\log n/\varepsilon)^f$ we had for the original FT-trees (that we still need to preprocess). The more efficient expath computation transfers to positive granularity, see the full version. We can compute them in time $\widetilde{O}(fm+\lambda) = \widetilde{O}(m)$. So the preprocessing time of the new trees is dominated by the one for the old trees. Also, the additional $\widetilde{O}(nL^{3f+o(1)})$ term for the sparse/dense balls will turn out to be negligible. More importantly, for the total size of the new trees the number of nodes gets multiplied by $O(\lambda + f\log^2(n)/\varepsilon)$, proportional to the number of parts. The result turns out to be

$$\widetilde{O}\left(\frac{n^2}{L^{f-1}}\right) + \widetilde{O}\left(\frac{n^2}{\lambda L^{2f-2}}\right)\cdot O\left(\frac{\log n}{\varepsilon}\right)^f + \widetilde{O}\left(\frac{n^2}{\varepsilon L^f}\right) + \widetilde{O}\left(\frac{n^2}{\lambda^2L^{2f-2}}\right)\cdot O\left(\frac{\log n}{\varepsilon}\right)^{f+1}.$$

With $f \geqslant 2$ (see Theorem 1.1), all the terms are at most the $\widetilde{O}(n^2/\lambda)\cdot O(\log n/\varepsilon)^{f+1}$ for the old FT-trees. Again, the $\widetilde{O}(nL^{2f+o(1)}/\lambda)$ space to store the regular pivots in the sparse balls will be irrelevant.

A straightforward generalization of Lemma 5.8 shows that evaluating $FT_\lambda(b_u,b_v)$ with query set F takes time $\widetilde{O}(L^{o(1)}(\lambda+1/\varepsilon))$.

Computing $w'_{H^F}(u,v)$. Let again $G_1,\dots,G_\kappa$ be the graphs in the leaves of the sampling trees (Section 4.2). For every G_i and vertex $x \in V$ for which $|\text{ball}_{G_i}(x,\lambda)| > L^f$, we said we store a

[12] This marker is made more precise in Section 6.2.
[13] If this holds for both u and v the choice of x is arbitrary.

marker. More precisely, we associate with (G_i, x) a *single* new pivot $b_x \in \mathcal{B} \cap \text{ball}_{G_i}(x, \lambda)$. As before, let $i_1, \dots, i_r$ be the indices of graphs G_{i_j} that are relevant for the query (u, v, F). Even if $\text{ball}_{G-F}(u, \lambda)$ and $\text{ball}_{G-F}(v, \lambda)$ are dense, it might be that all the $\text{ball}_{G_{i_j}}(u, \lambda)$ are sparse or all $\text{ball}_{G_{i_j}}(v, \lambda)$ are sparse. If so, we compute $w'_{H^F}(u, v)$ (and in turn $w_{H^F}(u, v)$) via Equation (1). Otherwise, there are indices $i_u, i_v \in \{i_1, \dots, i_r\}$ such that both $|\text{ball}_{G_{i_u}}(u, \lambda)| > L^f$ and $|\text{ball}_{G_{i_v}}(v, \lambda)| > L^f$. If there are multiple such indices, the choice is arbirary. Let $b_u \in \mathcal{B} \cap \text{ball}_{G_{i_u}}(u, \lambda)$ and let $b_v \in \mathcal{B} \cap \text{ball}_{G_{i_v}}(v, \lambda)$ be the stored new pivots. We define the auxiliary weight as

$$w'_{H^F}(u, v) = FT_\lambda(b_u, b_v, F) + 2\lambda. \tag{2}$$

This takes time $\widetilde{O}(L^{o(1)}(\lambda + 1/\varepsilon))$, much less than with sparse balls.

6.3 Approximation Guarantee

Towards Theorem 1.1, we already have that the space and preprocessing time is dominated by the original FT-trees, when accounting for $|B| = \widetilde{O}(n/\lambda)$. We also argued the query time. The plan to prove the approximation guarantee is the same as in Section 5, which crucially involved Lemma 5.6. We already discussed how to transfer its "first case", as well as the "second case" if both $\text{ball}_{G-F}(u, \lambda)$ and $\text{ball}_{G-F}(v, \lambda)$ are sparse. The "third case" is handled by Lemma 5.3 together with an induction. Due to the restricted space, we focus here on the secon "second case" also if the balls are dense, and how to adapt Lemma 5.6. As a first step, we generalize Lemmas 5.10 and 5.11 to FT-trees with granularity $\lambda > 0$.

Lemma 6.4. *Let $b_u, b_v \in \mathcal{B}$, P be any path between b_u and b_v, $e \in E(P)$, and y a vertex of the edge e. Then, $E(\text{seg}_\lambda(e, P)) = \{e\}$ or $|\text{seg}_\lambda(e, P)| \leqslant \frac{\varepsilon}{36}\big(\min(|P[b_u..y]|, |P[y..b_v]|) - \lambda\big)$.*

Recall that $d^{(\ell)}(u, v, A)$, for some $A \subseteq E$, is the length of the shortest ℓ-decomposable path in $G - A$.

Lemma 6.5. *Let $u, v \in V$ be two vertices, $A \subseteq E$ a set of edges, and $b_u \in \mathcal{B} \cap \text{ball}_{G-A}(u, \lambda)$ and $b_v \in \mathcal{B} \cap \text{ball}_{G-A}(u, \lambda)$. Let further ℓ be a positive integer, and P the shortest ℓ-expath with granularity λ between b_u and b_v in $G - A$. Then, for every $y \in V(P)$ with $|P[b_u..y]|, |P[y..b_v]| > \lambda$, it holds that $|P[b_u..y]| \leqslant 4 \cdot d^{(\ell)}(u, y, A) + \lambda$ and $|P[y..b_v]| \leqslant 4 \cdot d^{(\ell)}(y, v, A) + \lambda$.*

We use the results to show that also Lemma 5.9 transfers to non-vanishing granularity, but with a slight loss in the approximation. Again, $d^{(2f+1)}_{\varepsilon/9}(u, b, F)$ is the length of the shortest $(2f+1)$-decomposable u-v-path in $G - F$ that is *far away* from all failures.

Lemma 6.6. *Define $\delta = 8\lambda/L$. Let $u, v \in V$ be such that both $|\text{ball}_{G-F}(u, \lambda)|, |\text{ball}_{G-F}(u, \lambda)| > L^f$, and $b_u, b_v \in \mathcal{B}$ the associated new pivots. Let P be any $(2f+1)$-decomposable path between u and v in $G - F$ that is far away from F. Then, $d_{G-F}(u, v) \leqslant FT_\lambda(b_u, b_v, F) + 2\lambda \leqslant 3|P| + \delta L$. Moreover, if the* shortest *$(2f+1)$-decomposable path between u and v in $G - F$ that is far away from F has more than L edges, then, $FT_\lambda(b_u, b_v, F) + 2\lambda \leqslant (3 + \delta) \cdot d^{(2f+1)}_{\varepsilon/9}(u, v, F)$.*

Proof. We prove the survival of P all the way to the output node v^* of $FT_\lambda(b_u, b_v)$ when queried with set F, as in Lemma 5.9 We have to take care of the fact that P and P_{v^*} may have different endpoints. In fact, we argue about a longer path. Let $P(b_u, u, F)$ be the replacement path between b_u and u in G–F. $P(b_u, u, F)$ has at most λ edges by the choice $b_u \in \text{ball}_{G-F}(u, \lambda)$, same for $P(v, b_v, F)$. Also P is $(2f+1)$-decomposable, thus $Q = P(b_u, u, F) \circ P \circ P(v, b_v, F)$ is an $(2f+1)$-expath with granularity λ. We argue by induction that Q exists in the graph $G - A_v$ for every visited node v. This is clear for the root. For a non-output node $v \neq v^*$, let v' be its visited child.

To reach a contradiction, assume Q does not exist in $G - A_{v'}$ Thus, there is a segment of P_v that contains both a failing edge of F and an edge of Q. Without loosing generality, we choose $e_F \in F$ and $e_Q \in E(Q)$ such that both e_F and e_Q are in P_v and the subpath of P_v containing both edges contains no other failing edge. Let $y \in e_Q$ the endpoint closer to e_F along P_v, and let $z \in e_F$ the endpoint closer to e_Q. The subpath $P_v[y..z]$ is entirely in $G - F$.

It must be that $e_F \neq e_Q$ as Q lies in $G - F$. Segments with more than one edge only appear in the middle part of the stored expath, $|P_v[b_u..y]|, |P_v[y..b_v]| > \lambda$. By Lemmas 6.4 and 6.5, this means

$$\begin{aligned}
|\text{seg}_\lambda(e_F, P)| &\leqslant \frac{\varepsilon}{36}\Big(\min\big(|P[b_u..y]|, |P[y..b_v]|\big) - \lambda\Big) \\
&\leqslant \frac{\varepsilon}{36}\Big(\min\big(4d^{(2f+1)}(u, y, A_v) + \lambda, 4d^{(2f+1)}(y, v, A_v) + \lambda\big) - \lambda\Big) \\
&= \frac{\varepsilon}{9}\min\big(d^{(2f+1)}(u, y, A_v), d^{(2f+1)}(y, v, A_v)\big).
\end{aligned}$$

Subpaths of expaths with granularity are again expaths with granularity (the components P_a, P_c in Definition 6.3 can be empty). The subpath $P_v[b_u..y]$ (resp. $P_v[y..b_v]$) is the *shortest* $(2f+1)$-expath with granularity λ from b_u to y (from y to b_v) in G–A_v. By induction Q exists in $G - A_v$. $Q[b_u..y]$ (resp. $Q[y..b_v]$) is *some* $(2f+1)$-expath. Together with $|P_v[b_u..y]|, |P_v[y..b_v]| > \lambda$, this shows that y must also lie in the middle part of Q, that is, in P. Moreover P is some $(2f+1)$-decomposable path from u to v in $G - A_v$. In other words, $d^{(2f+1)}(u, y, A_v) \leqslant |P[u, y]|$ and $d^{(2f+1)}(y, v, A_v) \leqslant |P[y..v]|$.

Since P is far away from e_F, we have $|\text{seg}_\lambda(e_F, P)| \geqslant |P_v[y..z]| \geqslant d_{G-F}(y, z) > \frac{\varepsilon}{9}\min(|P[u..y]|, |P[y..b]|)$, a contradiction.

For the approximation, we prove $d_{G-F}(u, v) \leqslant FT_\lambda(b_u, b_v, F) + 2\lambda \leqslant (3+\delta) \cdot d^{(2f+1)}_{\varepsilon/9}(u, v, F)$ with $\delta = 8\lambda/L$ if P is the *shortest* $(2f+1)$-decomposable path from u to v in $G - F$ and has more than L edges. In particular, we have $|P| = d^{(2f+1)}_{\varepsilon/9}(u, v, F)$. The other claim is established in passing.

Recall that $FT_\lambda(b_u, b_v, F) = 3|P_{v^*}|$ for the output node v^*, for which we determined that $3|P_{v^*}| \geqslant d_{G-F}(b_u, b_v)$. By the triangle inequality, it holds that $FT_\lambda(b_u, b_v, F) + 2\lambda \geqslant d_{G-F}(b_u, b_v) + d_{G-F}(u, b_u) + d_{G-F}(b_v, v) \geqslant d_{G-F}(u, v)$. We have seen that Q survives until v^* and that P_{v^*} is not longer than Q. In summary, $FT_\lambda(b_u, b_v, F) + 2\lambda \leqslant 3|Q| + 2\lambda \leqslant 3(|P| + 2\lambda) + 2\lambda \leqslant 3|P| + 8\lambda = 3|P| + \delta L < (3 + \delta)\, d^{(2f+1)}_{\varepsilon/9}(u, v, F)$. □

Lastly, we only sketch the necessary changes to Lemma 5.6 to derive a $(3+\varepsilon)$-approximation. Recall that we assume $\Delta = 3 - \varepsilon > 0$ to be a constant. We define $\lambda = \frac{\Delta}{96}\varepsilon L$, which in turn implies $\delta = \frac{\Delta}{12}\varepsilon$. In fact, any $\delta \leqslant \frac{3-\varepsilon}{9+\varepsilon}\varepsilon$ would do as this ensures $\delta + (6 + \delta + \varepsilon)\frac{\varepsilon}{9} \leqslant \varepsilon$. In Lemma 5.6 we had $w_{H^F}(u, v) \leqslant 3\, d_{G-F}(u, v)$ if the path was short ("first case") or long but far away from all failures ("second case"). We now only have $w_{H^F}(u, v) \leqslant (3+\delta)\, d_{G-F}(u, v)$ due to the dense ball case. In the "third case", we use the the x-y-z-argument of Lemma 5.3. A similar reasoning as before gives $w_{H^F}(u, z) \leqslant (3+\delta)(1+\frac{\varepsilon}{9}) d_{G-F}(u, y)$ (instead of $(3+\frac{\varepsilon}{3})\, d_{G-F}(u, y)$). The crucial

part is the chain of inequalities at the end of the proof:

$$
\begin{aligned}
&d_{H^F}(u,v) \leqslant w_{H^F}(u,z) + d_{H^F}(z,v)\\
&\leqslant (3+\delta)\left(1+\frac{\varepsilon}{9}\right) d_{G-F}(u,y) + (3+\varepsilon) d_{G-F}(z,v)\\
&= (3+\delta)\left(1+\frac{\varepsilon}{9}\right) d_{G-F}(u,y) + (3+\varepsilon) d_{G-F}(z,y) + (3+\varepsilon) d_{G-F}(y,v)\\
&\leqslant (3+\delta)\left(1+\frac{\varepsilon}{9}\right) d_{G-F}(u,y) + (3+\varepsilon)\frac{\varepsilon}{9} d_{G-F}(u,y) + (3+\varepsilon) d_{G-F}(y,v)\\
&= 3d_{G-F}(u,y) + \delta \cdot d_{G-F}(u,y) + (6+\delta+\varepsilon)\frac{\varepsilon}{9} \cdot d_{G-F}(u,y) +\\
&\qquad (3+\varepsilon)\, d_{G-F}(y,v)\\
&\leqslant 3d_{G-F}(u,y) + \varepsilon \cdot d_{G-F}(u,y) + (3+\varepsilon)\, d_{G-F}(y,v)\\
&= (3+\varepsilon)\, d_{G-F}(u,v).
\end{aligned}
$$

The parameters of Theorem 1.1 follow from the discussion in this section and the choices $\lambda = \frac{\Delta}{96}\varepsilon L$ and $L = n^{\alpha/(f+1)}$ in the same fashion as in Lemma 5.1. The main difference is the transition from L to λ, giving an extra $1/\varepsilon$ factor in the space and preprocessing time, and (of course) the improved query time.

ACKNOWLEDGMENTS

The authors thank Merav Parter for raising the question of designing distance sensitivity oracles that require only subquadratic space.

This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program, grant agreement No. 803118 "The Power of Randomization in Uncertain Environments (UncertainENV)" and grant agreement No. 101019564 "The Design and Evaluation of Modern Fully Dynamic Data Structures (MoDynStruct)".

REFERENCES

[1] Yehuda Afek, Anat Bremler-Barr, Haim Kaplan, Edith Cohen, and Michael Merritt. 2002. Restoration by Path Concatenation: Fast Recovery of MPLS Paths. *Distributed Computing* 15 (2002), 273–283. https://doi.org/10.1007/s00446-002-0080-6

[2] Noga Alon, Shiri Chechik, and Sarel Cohen. 2019. Deterministic Combinatorial Replacement Paths and Distance Sensitivity Oracles. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming, (ICALP)*. 12:1–12:14. https://doi.org/10.4230/LIPIcs.ICALP.2019.12

[3] Surender Baswana and Neelesh Khanna. 2013. Approximate Shortest Paths Avoiding a Failed Vertex: Near Optimal Data Structures for Undirected Unweighted Graphs. *Algorithmica* 66 (2013), 18–50. https://doi.org/10.1007/s00453-012-9621-y

[4] Michael A. Bender and Martín Farach-Colton. 2000. The LCA Problem Revisited. In *Proceedings of the 4th Latin American Symposium Theoretical Informatics (LATIN)*. 88–94. https://doi.org/10.1007/10719839_9

[5] Aaron Bernstein and David R. Karger. 2008. Improved Distance Sensitivity Oracles via Random Sampling. In *Proceedings of the 19th Symposium on Discrete Algorithms (SODA)*. 34–43. https://dl.acm.org/doi/abs/10.5555/1347082.1347087

[6] Aaron Bernstein and David R. Karger. 2009. A Nearly Optimal Oracle for Avoiding Failed Vertices and Edges. In *Proceedings of the 41st Symposium on Theory of Computing (STOC)*. 101–110. https://doi.org/10.1145/1536414.1536431

[7] Davide Bilò, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, and Martin Schirneck. 2022. Deterministic Sensitivity Oracles for Diameter, Eccentricities and All Pairs Distances. In *Proceedings of the 49th International Colloquium on Automata, Languages, and Programming (ICALP)*. 22:1–22:19. https://doi.org/10.4230/LIPIcs.ICALP.2022.22

[8] Davide Bilò, Sarel Cohen, Tobias Friedrich, and Martin Schirneck. 2021. Near-Optimal Deterministic Single-Source Distance Sensitivity Oracles. In *Proceedings of the 29th European Symposium on Algorithms (ESA)*. 18:1–18:17. https://doi.org/10.4230/LIPIcs.ESA.2021.18

[9] Davide Bilò, Sarel Cohen, Tobias Friedrich, and Martin Schirneck. 2021. Space-Efficient Fault-Tolerant Diameter Oracles. In *Proceedings of the 46th International Symposium on Mathematical Foundations of Computer Science (MFCS)*. 18:1–18:16. https://doi.org/10.4230/LIPIcs.MFCS.2021.18

[10] Greg Bodwin, Michael Dinitz, and Caleb Robelle. 2021. Optimal Vertex Fault-Tolerant Spanners in Polynomial Time. In *Proceedings of the 32nd Symposium on Discrete Algorithms (SODA)*. 2924–2938. https://doi.org/10.1137/1.9781611976465.174

[11] Greg Bodwin, Michael Dinitz, and Caleb Robelle. 2022. Partially Optimal Edge Fault-Tolerant Spanners. In *Proceedings of the 33rd Symposium on Discrete Algorithms (SODA)*. 3272–3286. https://doi.org/10.1137/1.9781611977073.129

[12] Sergio Cabello, Erin W. Chambers, and Jeff Erickson. 2013. Multiple-Source Shortest Paths in Embedded Graphs. *SIAM J. Comput.* 42 (2013), 1542–1571. https://doi.org/10.1137/120864271

[13] Shiri Chechik and Sarel Cohen. 2020. Distance Sensitivity Oracles with Subcubic Preprocessing Time and Fast Query Time. In *Proccedings of the 52nd Symposium on Theory of Computing (STOC)*. 1375–1388. https://doi.org/10.1145/3357713.3384253

[14] Shiri Chechik, Sarel Cohen, Amos Fiat, and Haim Kaplan. 2017. $(1+\varepsilon)$-Approximate f-Sensitive Distance Oracles. In *Proceedings of the 28th Symposium on Discrete Algorithms (SODA)*. 1479–1496. https://doi.org/10.1137/1.9781611974782.96

[15] Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. 2010. Fault Tolerant Spanners for General Graphs. *SIAM J. Comput.* 39 (2010), 3403–3423. https://doi.org/10.1137/090758039

[16] Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. 2012. f-Sensitivity Distance Oracles and Routing Schemes. *Algorithmica* 63 (2012), 861–882. https://doi.org/10.1007/s00453-011-9543-0

[17] Camil Demetrescu and Mikkel Thorup. 2002. Oracles for Distances Avoiding a Link-Failure. In *Proceedings of the 13th Symposium on Discrete Algorithms (SODA)*. 838–843. https://dl.acm.org/doi/10.5555/545381.545490

[18] Camil Demetrescu, Mikkel Thorup, Rezaul A. Chowdhury, and Vijaya Ramachandran. 2008. Oracles for Distances Avoiding a Failed Node or Link. *SIAM J. Comput.* 37 (2008), 1299–1318. https://doi.org/10.1137/S0097539705429847

[19] Ran Duan and Seth Pettie. 2009. Dual-Failure Distance and Connectivity Oracles. In *Proceedings of the 20th Symposium on Discrete Algorithms (SODA)*. 506–515. https://dl.acm.org/doi/10.5555/545381.545490

[20] Ran Duan and Hanlin Ren. 2022. Maintaining Exact Distances under Multiple Edge Failures. In *Proceedings of the 54th Symposium on Theory of Computing (STOC)*. 1093–1101. https://doi.org/10.1145/3519935.3520002

[21] Paul Erdős. 1964. Extremal Problems in Graph Theory. *Theory of Graphs and its Applications* (1964), 29–36.

[22] Fabrizio Grandoni and Virginia Vassilevska Williams. 2020. Faster Replacement Paths and Distance Sensitivity Oracles. *ACM Transaction on Algorithms* 16 (2020), 15:1–15:25. https://doi.org/10.1145/3365835

[23] Yong Gu and Hanlin Ren. 2021. Constructing a Distance Sensitivity Oracle in $O(n^{2.5794}M)$ Time. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP)*. 76:1–76:20. https://doi.org/10.4230/LIPIcs.ICALP.2021.76

[24] Torben Hagerup, Peter Bro Miltersen, and Rasmus Pagh. 2001. Deterministic Dictionaries. *Journal of Algorithms* 41 (2001), 69–85. https://doi.org/10.1006/jagm.2001.1171

[25] Karthik C.S. and Merav Parter. 2021. Deterministic Replacement Path Covering. In *Proceedings of the 32nd Symposium on Discrete Algorithms (SODA)*. 704–723. https://doi.org/10.1137/1.9781611976465.44

[26] Christos Levcopoulos, Giri Narasimhan, and Michiel H. M. Smid. 1998. Efficient Algorithms for Constructing Fault-Tolerant Geometric Spanners. In *Proceedings of the 30th Symposium on Theory of Computing (STOC)*. 186–195. https://doi.org/10.1145/276698.276734

[27] Hanlin Ren. 2022. Improved Distance Sensitivity Oracles with Subcubic Preprocessing Time. *J. Comput. System Sci.* 123 (2022), 159–170. https://doi.org/10.1016/j.jcss.2021.08.005

[28] Liam Roditty and Uri Zwick. 2012. Replacement Paths and k Simple Shortest Paths in Unweighted Directed Graphs. *ACM Transaction on Algorithms* 8, Article 33 (2012), 33:1–33:11 pages. https://doi.org/10.1145/2344422.2344423

[29] Mikkel Thorup and Uri Zwick. 2005. Approximate Distance Oracles. *J. ACM* 52 (2005), 1–24. https://doi.org/10.1145/1044731.1044732

[30] Oren Weimann and Raphael Yuster. 2013. Replacement Paths and Distance Sensitivity Oracles via Fast Matrix Multiplication. *ACM Transactions on Algorithms* 9 (2013), 14:1–14:13. https://doi.org/10.1145/2438645.2438646

Received 2022-11-07; accepted 2023-02-06