Improved Distance (Sensitivity) Oracles with Subquadratic Space

Davide Bilò[1], Shiri Chechik[2], Keerti Choudhary[3],
Sarel Cohen[4], Tobias Friedrich[5], and Martin Schirneck[6]

[1]Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila
davide.bilo@univaq.it

[2]Department of Computer Science, Tel Aviv University
shiri.chechik@gmail.com

[3]Department of Computer Science and Engineering, Indian Institute of Technology Delhi,
keerti@iitd.ac.in

[4]School of Computer Science, The Academic College of Tel Aviv-Yaffo
sarelco@mta.ac.il

[5]Hasso Plattner Institute, University of Potsdam
tobias.friedrich@hpi.de

[6]Faculty of Computer Science, University of Vienna
martin.schirneck@univie.ac.at

Abstract

A *distance oracle* (DO) for a graph G is a data structure that, when queried with vertices s, t, returns an estimate $\widehat{d}(s,t)$ of their distance in G. The oracle has stretch (α, β) if the estimate satisfies $d(s,t) \leqslant \widehat{d}(s,t) \leqslant \alpha \cdot d(s,t) + \beta$. An *$f$-edge fault-tolerant distance sensitivity oracle* (f-DSO) additionally receives a set F of up to f edges and estimates the distance in $G-F$.

Our first contribution is the design of new distance oracles with subquadratic space for undirected graphs. We show that introducing a small additive stretch $\beta > 0$ allows one to make the multiplicative stretch α arbitrarily small. This sidesteps a known lower bound of $\alpha \geqslant 3$ (for $\beta = 0$ and subquadratic space) [Thorup & Zwick, JACM 2005]. We present a DO for graphs with edge weights in $[0, W]$ that, for any positive integer ℓ and any $c \in (0, \ell/2]$, has stretch $(1+\frac{1}{\ell}, 2W)$, space $\widetilde{O}(n^{2-\frac{c}{\ell}})$, and query time $O(n^c)$. These are the first subquadratic-space DOs with $(1+\varepsilon, O(1))$-stretch generalizing Agarwal and Godfrey's results for sparse graphs [SODA 2013] to general undirected graphs. We also construct alternative DOs with even smaller space at the cost of a higher additive stretch. For any integer $k \geqslant 1$, the DOs have a stretch $(2k-1+\frac{1}{\ell}, 4kW)$, space $O(n^{1+\frac{1}{k}(1-\frac{c}{8\ell})})$, and query time $O(n^c)$.

Our second contribution is a framework that turns any (α, β)-stretch DO for unweighted graphs into an $(\alpha(1+\varepsilon), \beta)$-stretch f-DSO with sensitivity $f = o(\log(n)/\log\log n)$ and retains subquadratic space. This generalizes a result by Bilò, Chechik, Choudhary, Cohen, Friedrich, Krogmann, and Schirneck [STOC 2023, TheoretiCS 2024] for the special case of stretch $(3, 0)$ and $f = O(1)$. We also derandomize the entire construction. By combining the framework with our new distance oracle, we obtain an f-DSO that, for any $\gamma \in (0, (\ell+1)/2]$, has stretch $((1+\frac{1}{\ell})(1+\varepsilon), 2)$, space $n^{2-\frac{\gamma}{(\ell+1)(f+1)}+o(1)}/\varepsilon^{f+2}$, and query time $\widetilde{O}(n^\gamma/\varepsilon^2)$. This is the first deterministic f-DSO with subquadratic space, near-additive stretch, and sublinear query time.

1 Introduction

A *distance oracle* (DO) is a data structure to retrieve (exact or approximate) distances between any pair of vertices s, t in an undirected graph $G = (V, E)$ upon query. The problem of designing distance oracles has attracted a lot of attention in recent years due to the wide applicability in domains like network routing, traffic engineering, and distributed computing. These oracles are used in settings where one cannot afford to store the entire graph, but still wants to be able to quickly query graph distances. A DO has *stretch* (α, β) if, for any pair s and t, the value $\widehat{d}(s,t)$ returned by the DO satisfies $d(s,t) \leqslant \widehat{d}(s,t) \leqslant \alpha \cdot d(s,t) + \beta$, where $d(s,t)$ denotes the exact distance between s and t in G. As networks in most real-life applications are prone to transient failures, researchers have also studied the problem of designing oracles that additionally tolerate multiple edge failures in G. An *f-edge fault-tolerant distance sensitivity oracle* (f-DSO) with stretch (α, β) is a data structure that, when queried on a triple (s, t, F), where $F \subseteq E$ has size at most f, outputs an estimate $\widehat{d}(s,t,F)$ of the distance $d(s,t,F)$ from s to t in $G-F$ such that $d(s,t,F) \leqslant \widehat{d}(s,t,F) \leqslant \alpha \cdot d(s,t,F) + \beta$.

Several DOs and f-DSOs with different size-stretch-time trade-offs have been developed in the last decades. See, for example, [Aga14, BGS09, CC20, CCFK17, DG24, DP09, DR22, DT02, GV20, PR14, Ren22, TZ05, WY13]. Our focus is on providing new distance oracles with a subquadratic space usage for both static and error-prone graphs. A special focus of this work is how static distance oracles can be converted into fault-tolerant distance *sensitivity* oracles.

1.1 Approximate Distance Oracles for Static Graphs

We first discuss distance oracles. Extensive research has been dedicated in that direction in the past two decades. In their seminal paper [TZ05], Thorup and Zwick showed that, for any positive integer k (possibly depending on the input size), an undirected graph with n vertices and m edges can be preprocessed in time $O(mn^{1/k})$ to obtain an oracle with multiplicative stretch $2k-1$, space[1] $O(kn^{1+1/k})$, and query time $O(k)$. Subsequent works [Che14, Che15, WN13] further improved the space of the construction to $O(n^{1+1/k})$ and the query time to $O(1)$.

In the case of $k = 2$, this results in a 3-approximate oracle taking subquadratic space $O(n^{3/2})$. A simple information theoretic lower bound using bipartite graphs [TZ05] shows that distance oracles with a purely multiplicative stretch *below* 3 require space that is at least quadratic in n. Pătraşcu and Roditty [PR14] were arguably the first to introduce an additive stretch to simultaneously reduce the space and the multiplicative stretch in general dense graphs. They proposed a distance oracle for unweighted graphs with stretch $(2, 1)$ that takes $O(n^{5/3})$ space, has a a constant query time and can be constructed in time $O(mn^{2/3})$. They also showed that DOs with multiplicative stretch $\alpha \leqslant 2$ and constant query time require $\Omega(n^2)$ space, assuming a conjecture on the hardness of set intersection queries.

Agarwal and Godfrey [AG13] studied $(1+\varepsilon, O(1))$-stretch DOs for sparse graphs. However, when transferred to dense graphs the space of their construction becomes $\Omega(n^2)$ and the query time is $\Omega(n)$. To the best of our knowledge, no distance oracles have been constructed that simultaneously have subquadratic space and a multiplicative stretch better than 2 for general undirected graphs. If we want to reduce α while retaining low space, we necessarily have to introduce some additive stretch β and the query time must rise beyond constant. This raises the following natural question.

[1] The space of the data structures is measured in the number of machine words on $O(\log n)$ bits.

Question. *Is there a distance oracle with a $1+\varepsilon$ multiplicative and constant additive stretch that takes subquadratic space and has a query time that is sublinear in n?*

We provide an affirmative answer by presenting the first oracle with stretch $(1{+}\varepsilon, O(1))$ for general graphs. The construction is surprisingly simple, has subquadratic space, and sublinear query time, improving over Agarwal and Godfrey's work for dense graphs.

Theorem 1. *Let W be a non-negative real number, and G an undirected graph with n vertices and edge weights in a $\mathsf{poly}(n)$-sized subset of $[0, W]$. For every positive integer $K \leqslant \sqrt{n}$ and any $\varepsilon > 0$, there exists a path-reporting distance oracle for G that has stretch $(1{+}\varepsilon, 2W)$, space $\widetilde{O}(n^2/K)$, and query time $O(K^{\lceil 1/\varepsilon \rceil})$ for the distance and an additional $O(1)$ time per reported edge. The data structure can be constructed in APSP time.*

The restriction of the edge weights to a polynomial-sized subset of $[0, W]$ is to ensure that any graph distance can be encoded in a constant number of $O(\log n)$-bit words. We remark that our construction in fact guarantees a stretch of $(1{+}\varepsilon, 2w_{s,t})$ where $w_{s,t}$ is the maximum edge weight along a shortest path from s to t in G. The stretch thus depend locally on the queried vertices rather than the global maximum edge weight.

One cannot reduce the additive stretch in Theorem 1 to 1 (if $\varepsilon < 2$) even in unweighted graphs as the unconditional lower bound for bipartite graphs [TZ05] stated earlier rules out any data structure that can distinguish between distances 1 and 3 in subquadratic space.

By setting $\varepsilon = 1/\ell$ and $K^\ell = n^c$, we get a trade-off between stretch space and time.

Corollary 2. *For every positive integer t and any $0 < c \leqslant \ell/2$, there exists a distance oracle for undirected graphs with stretch $(1{+}\frac{1}{\ell}, 2W)$, space $\widetilde{O}(n^{2-\frac{c}{\ell}})$, and query time $O(n^c)$.*

An extension of our construction allows to trade a higher stretch for a lower space. Namely, we present a family of DOs with multiplicative stretch of $2k{-}1+\varepsilon$ and $o(n^{1+1/k})$ space.

Theorem 3. *Let W be a non-negative real number, and G an undirected graph with n vertices and edge weights in a $\mathsf{poly}(n)$-sized subset of $[0, W]$. For all positive integers k and K with $K = O(n^{1/(2k+1)})$, and every $\varepsilon > 0$, there exits a distance oracle for G that has stretch $(2k{-}1{+}\varepsilon, 4kW)$, space $O((\frac{n}{K})^{1+1/k} \log^{1+1/k} n)$, and query time $O(K^{2\lceil 4k/\varepsilon \rceil})$. The data structure can be constructed in APSP time.*

For $k = 1$, the oracle in Theorem 3 has the same multiplicative stretch of $1+\varepsilon$ as the one in Theorem 1 and a better space of $\widetilde{O}(n^2/K^2)$, but the additive stretch of $4W$ is larger and so is the query time. We obtain the following corollary for general k, $\varepsilon = 1/\ell$, and $K^{8k\ell} = n^c$.

Corollary 4. *For all positive integers k and ℓ, and any $0 < c \leqslant (4{-}\frac{4}{2k+1})\,\ell$, there exists a distance oracle with stretch $(2k{-}1{+}\frac{1}{\ell}, 4kW)$, space $\widetilde{O}(n^{1+\frac{1}{k}(1-\frac{c}{8\ell})})$, and query time $O(n^c)$.*

Probably closest to hierarchy we present in Theorem 3 is the distance labeling scheme of Abraham and Gavoille [AG11]. Seen as a DO for unweighted graphs, for any integer $k \geqslant 2$, it has a stretch of $(2k{-}2, 1)$ space $\widetilde{O}(n^{1+\frac{2}{2k-1}})$, and query time $O(k)$.

1.2 Distance Sensitivity Oracles

Most of the proposed distance sensitivity oracles that treat the sensitivity f (the number of tolerated edge failures) as a parameter require $\Omega(n^2)$ space, have a stretch depending on f, or an $\Omega(n)$ query time. See Section 1.3 for more details. Bilò, Chechik, Choudhary, Cohen, Friedrich, Krogmann,

and Schirneck [BCC+24] were the first to introduce an f-DSO with subquadratic space, a constant multiplicative stretch of $3+\varepsilon$ (for any f), and a query time that can be made an arbitrarily small polynomial. More precisely, for any unweighted graph G with unique shortest paths, every integer constant $f \geqslant 2$, any $0 < \gamma < 1/2$, and $\varepsilon > 0$, they devised an f-DSO for G with stretch $3+\varepsilon$, space $\widetilde{O}(n^{2-\frac{\gamma}{f+1}}) \cdot O(\log n/\varepsilon)^{f+2}$, query time $O(n^{\gamma}/\varepsilon^2)$, and preprocessing time $\widetilde{O}(mn^{2-\frac{\gamma}{f+1}}) \cdot O(\log n/\varepsilon)^{f+1}$. Even more than in the case of static distance oracles, a multiplicative stretch better than 3 (let alone close to 1) remains a barrier for subquadratic-space f-DSOs. We explore whether the introduction of a small additive stretch can help here as well.

Question. *Is there a distance sensitivity oracle for general sensitivity f with a $1+\varepsilon$ multiplicative stretch, possibly at the expense of a constant additive stretch, that takes subquadratic space and has a query time that is sublinear in n?*

We also answer this question affirmatively in this paper by presenting an f-DSO with stretch $(1+\varepsilon, 2)$ while still having subquadratic space and small polynomial query time.

Theorem 5. *Let ℓ be a positive integer constant and G be an undirected and unweighted graph with n vertices and m edges and unique shortest paths. For any $0 < \gamma \leqslant (\ell+1)/2$, sensitivity $2 \leqslant f = o(\log(n)/\log\log n)$, and approximation parameter $\varepsilon = \omega(\sqrt{\log(n)}/n^{\frac{\gamma}{2(\ell+1)(f+1)}})$, there exists an f-edge fault-tolerant distance sensitivity oracle for G that has*

- *stretch* $((1+\frac{1}{\ell})(1+\varepsilon), 2)$,
- *space* $n^{2-\frac{\gamma}{(\ell+1)(f+1)}+o(1)}/\varepsilon^{f+2}$,
- *query time* $O(n^{\gamma}/\varepsilon^2)$,
- *preprocessing time* $n^{2+\gamma+o(1)} + mn^{2-\frac{\gamma}{(\ell+1)(f+1)}+o(1)}/\varepsilon^{f+1}$.

We observe that the additive stretch of $\beta = 2$ is necessary due to the unconditional lower bound of $\Omega(n^2)$ on the size of every distance oracle with a purely multiplicative stretch of $\alpha < 3$, as discussed in Section 1.1. The assumption of unique shortest paths can be achieved, for example, by perturbing the edge weights with random small values. This means that an unweighted graph G becomes weighted and its edge weights are very close to 1. As an alternative, we can compute, in time $O(mn + n^2 \log^2 n)$, a set of unique shortest paths via *lexicographic pertubation* [CCE13].

Our main technique to obtaain the new distance sensitivity oracle is to develop a reduction that, given a path-reporting DO with stretch (α, β), constructs a $((1+\varepsilon)\alpha, \beta)$-stretch f-DSO. Crucially, the reduction results in a subquadratic-space f-DSO provided that the initial DO also takes only subquadratic space. Although the problem of designing static as well as fault-tolerant distance (sensitivity) oracles has been studied extensively in the past two decades, there has been no substantial progress to obtain black-box conversions from compact DOs to compact f-DSOs. The work by Bilò et al. [BCC+24] comes close, but their construction of the f-DSO is entangled with the inner workings of the DO of Thorup and Zwick [TZ05]. Also, their analysis relies heavily on the input distance oracle having a purely multiplicative stretch of 3. In contrast, we develop algorithms that can work with any distance oracle as long as it is able to report an approximate shortest path that adheres to the stretch bound. Our analysis is able to incorporate any multiplicative stretch $\alpha \geqslant 1$ as well as additive stretch β that satisfies very mild technical assumptions. (See Theorem 7 for the precise statement.) Plugging in the distance oracle with stretch $(1+\frac{1}{\ell}, 2)$ from Corollary 2 then allows us to achieve the $((1+\frac{1}{\ell})(1+\varepsilon), 2)$-stretch f-DSO with subquadratic space and small polynomial query time from Theorem 5.

Emulating Searches in Fault-Tolerant Trees. Our transformation that makes distance oracles fault tolerant has two major steps. In the first one, we only treat *hop-short* replacement paths, these are shortest paths in $G-F$ whose number of edges is bounded by some cut-off parameter L. We transform any (α, β)-stretch distance oracle into an f-distance sensitivity oracle for hop-short paths (f-DSO$^{\leqslant L}$) with the same stretch. The second step then combines the solutions for hop-short paths into a general distance sensitivity oracle (f-DSO).

For the the first step, we develop *fault-tolerant tree oracles*, which are a new way to retrieve hop-short replacement paths from a previously known data structure called fault-tolerant trees (FT-trees), but without actually storing those trees. FT-trees were originally introduced by Chechik, Cohen, Fiat, and Kaplan [CCFK17]. There is an FT-tree $FT(s,t)$ required for every pair of vertices and a query traverses along a root-to-leaf path in the respective tree, until a shortest s-t-path in $G-F$ is found. This solution uses super-quadratic $\Omega(n^2L^f)$ space in total and is thus too large for our purpose. We devise a technique to emulate a search in an FT-tree without access to the tree. We use those searches to generate a carefully chosen family of subgraphs of G and apply the input distance oracle to each of them. This ensures that any query (s,t,F) that satisfies $d(s,t,F) \leqslant L$, is answered with an (α, β)-approximate path.

Theorem 6. *Let G be an unweighted (possibly directed) graph, with n vertices, and let f and L be positive integer parameters possibly depending on n. Assume access to a path-reporting distance oracle that, on any spanning subgraph of G, has stretch (α, β), takes space S, query time Q, and preprocessing time T. Then, there is an f-edge fault-tolerant distance sensitivity oracle for replacement paths in G with at most L edges that has*

- *stretch (α, β),*
- *space $O(fL\log n)^f \cdot \mathsf{S}$,*
- *query time $\widetilde{O}(f \cdot (\mathsf{Q} + \alpha L + \beta + f^2 L))$,*
- *preprocessing time $\widetilde{O}(n^2(\alpha L+\beta)^f) \cdot (O(fL\log n)^f + \mathsf{Q}) + O(fL\log n)^f \cdot \mathsf{T}$.*

Note that the behavior of f-DSOs (hop-short or general) are usually only discussed for valid queries, that is, triples (s,t,F) where the edges in F are actually present in G. *Checking* for validity requires $\Omega(n^2)$ space, which would make subquadratic-space f-DSO impossible. Our query algorithm never uses the assumption $F \subseteq E$ and the data structure can be queried with any triplet (s,t,F) where $F \subseteq \binom{V}{2}$ is a set of at most f pairs of vertices. In this case it returns the approximate distance for the valid query $(s,t,F \cap E)$.[2]

General Distance Sensitivity Oracles. Bilò et al. [BCC+24] described how to apply an f-DSO$^{\leqslant L}$ (with the restriction to hop-short paths) to construct a general distance sensitivity oracle without the constraint. They used two sets of pivots, which are vertices at with the hop-short solutions are combined. However, their approach only works for f-DSO$^{\leqslant L}$ with purely multiplicative stretch of 3, resulting in an f-DSO with multiplicative stretch $3+\varepsilon$. We generalize this by introducing a new data structure called *pivot trees*, as well as pinpointing the places in their construction that need to be adapted to accompany multiplicative stretch $\alpha \neq 3$ an additive stretch $\beta > 0$. The new pivot trees allow us to quickly find relevant pivots in $G-F$ that are close enough to the endpoints s and t in the query. An additional difference to [BCC+24] is a more involved analysis of the stretch. The issue is that the additive part β accumulates while the answers of the f-DSO$^{\leqslant L}$ are aggregated. Fortunately, this happens only if the paths in questions are hop-long, that is, if

[2] See Lemmas 14 and 20 for the technical details.

they have more than L edges. This allows us to introduce an inductive argument controlling the stretch accumulation and charge the overhead to the multiplicative part instead. As a result, we are able to turn a f-DSO$^{\leqslant L}$ with stretch (α, β) into a general f-DSO that has stretch $(\alpha(1+\varepsilon), \beta)$, all this while keeping the total space subquadratic.

To simplify the presentation, we first provide a randomized solution in Section 6 and then show how to derandomize it in Section 7. For the randomized construction, the guarantees hold *with high probability* (w.h.p.), which we define as with probability at least $1 - n^{-c}$ for some constant $c > 0$. In fact, c can be made arbitrarily large without affecting the asymptotics. We later show how to derandomize the oracle without any loss in its features, under very mild assumption on the parameters f and L. The main source of randomization is the creation of the pivot sets. The aforementioned pivot trees not only allow us to get better bounds for the randomized data structure but even help with derandomizing it.

To the best of our knowledge, we provide the first deterministic f-DSO with subquadratic space, near-additive stretch, and sublinear query time.

Theorem 7. *Let G be an undirected, unweighted graph with n vertices, m edges, and unique shortest paths. Let f and L be positive integer parameters possibly depending on n and m such that $2 \leqslant f \leqslant L \leqslant n$ as well as $L = \omega(\log n)$. Assume access to an f-edge fault-tolerant distance sensitivity oracle for replacement paths in G with at most L edges, that has stretch (α, β), space S_L, query time Q_L, and preprocessing time T_L. Then, for every $\varepsilon = \varepsilon(n, m, f, L) > 0$, and $\beta = o\left(\frac{\varepsilon^2 L}{f^3 \log n}\right)$, there is a randomized (general) f-edge fault-tolerant distance sensitivity oracle for G that with high probability has*

- *stretch $(\alpha(1+\varepsilon), \beta)$,*
- *query time $\widetilde{O}(f^5 L^{f-1}(\mathsf{Q}_L + f)/\varepsilon^2)$,*
- *preprocessing time $\mathsf{T}_L + O(L^{3f} n) + \widetilde{O}(f^2 m n^2/L) \cdot O(\log n/\varepsilon)^{f+1}$.*

The space of the data structure is w.h.p.

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

If additionally $f \geqslant 4$ and $L = \widetilde{O}(\sqrt{f^3 m/\varepsilon})$, the data structure can be made deterministic with the same stretch, query time, preprocessing time, and space.

If $\widetilde{O}(f L^{2f}) = \widetilde{O}(n)$, the space in Theorem 7 simplifies to $\mathsf{S}_L + \widetilde{O}(f^2 n^2/L) \cdot O(\log n/\varepsilon)^{f+2}$. That means our construction has subquadratic space as long as the space requirement S_L of the input f-DSO for short hop-distances is subquadratic. We would like to add some context to the restriction on the additive stretch β. Assume for simplicity that f is a constant. In most cases in the literature where an f-DSO$^{\leqslant L}$ is used to build an f-DSO, L is of order $n^{\Theta(1/f)}$. We also use such a value in this work. In this case, our construction supports an additive stretch that can be as large as a small polynomial, as long as it is asymptotically smaller than $\varepsilon^2 n^{O(1/f)}$. Conversely, for a fixed β, the restriction can be interpreted as bounding how fast the approximation parameter ε can approach 0 (we do not require ε to be constant). In Theorem 5, we have $\beta = 2$, hence $\varepsilon = \omega(\sqrt{\log(n)}/n^{O(1/f)})$. Finally, the derandomization requires $f \geqslant 4$ and $L = \widetilde{O}(\sqrt{f^3 m/\varepsilon})$. Even in the unfavorable case in which both f and ε are constants, this allows for a cut-off parameter up to $\widetilde{O}(\sqrt{m})$. In very dense graphs, this is no restriction at all.

1.3 Related Work

Distance Oracles. Thorup and Zwick [TZ05] showed that, for any positive integer k, any DO for undirected graphs with a multiplicative stretch strictly less than $2k+1$ must take $\Omega(n^{1+1/k})$ bits off space, assuming the Erdős girth conjecture [Erd64]. The lower bound only applies to graphs that are sufficiently dense and to queries that involve pairs of neighboring vertices, leading to several attempts to bypass it in different settings. For example, there is a line of work on improved distance oracles for sparse graphs. Porat and Roditty [PR11] showed that for unweighted graphs and any $\varepsilon > 0$, one can construct a DO with multiplicative stretch $1+\varepsilon$ and query time $\widetilde{O}(m^{1-\frac{\varepsilon}{4+2\varepsilon}})$. The space of the data structure is $O(nm^{1-\frac{\varepsilon}{4+2\varepsilon}})$, which is subquadratic for $m = o(n^{1+\frac{\varepsilon}{4+\varepsilon}})$. Pătraşcu, Roditty, and Thorup [PRT12] obtained a series of DOs with fractional multiplicative stretches for sparse graphs. For general dense graphs, Pătraşcu and Roditty [PR14] devised a distance oracle for unweighted graphs with stretch $(2,1)$ that has $O(1)$ query time, $O(n^{5/3})$ space, and can be constructed in time $O(mn^{2/3})$. They also showed that (α,β)-approximate DOs with $2\alpha+\beta < 4$ require $\Omega(n^2)$ space, assuming conjecture on the hardness of set intersection queries. Compared to the Pătraşcu and Roditty upper bound [PR14], Baswana, Goyal, and Sen [BGS09] marginally increased the stretch to $(2,3)$ and space to $\widetilde{O}(n^{5/3})$ in order to reduce the preprocessing time to $\widetilde{O}(n^2)$. The stretch was later reset again to $(2,1)$ by Sommer [Som16], keeping the improved time complexity. A successive work by Knudsen [Knu17] removed all additional poly-logarithmic factors in both the construction time and space.

Agarwal and Godfrey [AG13, Aga14] investigated the possibility of constructing a distance oracle with a stretch less than 2, albeit at the expense of slower query times. They showed that, for any positive integer ℓ and any real number $c \in (0,1]$, it is possible to design a DO of size $O(m+n^{2-c})$ and multiplicative stretch $1+\frac{1}{\ell}$. The query time is $O((n^c\mu)^\ell)$, where $\mu = \frac{2m}{n}$ is the average degree of the graph. Furthermore, they also showed that the query time can be reduced to $O((n^c+\mu)^{2\ell-1})$ at the cost of a small additive stretch $\frac{2\ell-1}{t}W$, with Wbeing the maximum edge weight. Though the constructions in [AG13, Aga14] have a multiplicative stretch better than 2, their DOs have two main drawbacks. The subquadratic space only holds for sparse graphs, and, while they achieve a very low stretch, the query time is super-linear for dense graphs.

Akav and Roditty [AR20] proposed, for any $\varepsilon \in (0,\frac{1}{2})$, an $O(m+n^{2-\Omega(\varepsilon)})$-time algorithm that computes a DO with stretch $(2+\varepsilon,5)$ and $O(n^{11/6})$ space, thus breaking the quadratic time barrier for multiplicative stretch below 3. Chechik and Zhang [CZ22] improved this by offering both a DO with stretch $(2,3)$ that can be built in $\widetilde{O}(m+n^{1.987})$ time and a DO with stretch $(2+\varepsilon, c(\varepsilon))$ that can be built in $O(m+n^{\frac{5}{3}-\varepsilon})$ time, where $c(\varepsilon)$ is exponential in $1/\varepsilon$. Both data structures have space $\widetilde{O}(n^{5/3})$ and a constant query time.

Distance Sensitivity Oracles. Most of the work on distance sensitivity oracles is about handling a very small number $f \in \{1,2\}$ of failures [DT02, BK08, DTCR08, DP09, BK09, BK13, CC20, BCFS21, GV20, GR21, Ren22]. Here, we focus on related work with sensitivity $f \geqslant 3$ as this is the setting of the second problem in this paper. In their seminal work, Weimann and Yuster [WY13] designed a randomized f-DSO for exact distances introducing a size-time trade-off that is controlled by a parameter $\alpha \in (0,1)$. More precisely, their oracle w.h.p. has space $\widetilde{O}(n^{3-\alpha})$, query time of $\widetilde{O}(n^{2-2(1-\alpha)/f})$, and can be built in time $\widetilde{O}(mn^{2-\alpha})$. Van den Brand and Saranurak [BS19] and Karczmarz and Sankowski [KS23] presented f-DSOs using algebraic algorithms. However, their space requirement is at least quadratic and their query time is at least linear. Duan and Ren [DR22] provided an alternative f-DSO for exact distances with $O(fn^4)$ space, $f^{O(f)}$ query time, that is, constant whenever f is a constant. The preprocessing time for building their oracle is exponential in f, namely, $n^{\Omega(f)}$. Recently, Dey and Gupta [DG24] developed an f-DSO for undirected graphs

where each edge has an integral weight from $\{1 \dots W\}$ with $O((cf \log(nW))^{O(f^2)})$ query time, where $c > 1$ is some constant. It has near-quadratic space $O(f^4 n^2 \log^2(nW))$. A drawback of their oracles is again the preprocessing time of $\Omega(n^f)$, and their oracle requires at least $\Omega(n^2)$ space.

When allowing approximation, the f-DSO of Chechik et al. [CCFK17] guarantees a multiplicative stretch of $1+\varepsilon$ with a space requirement of $O(n^{2+o(1)} \log W)$, where $\varepsilon > 0$ W is constant and W is the weight of the heaviest edge. Their oracle can handle up to $f = o(\log n / \log \log n)$ failures, has a query time of $O(f^5 \log n \log \log W)$, and can be build in $O(n^{5+o(1)}(\log W)/\varepsilon^f)$ time. In fact, the preprocessing time has recently been reduced to $O(mn^{2+o(1)}/\varepsilon^f)$ [BCC+24]

Besides the general Thorup-Zwick bound [TZ05] that assumes the girth conjecture, they also showed *unconditionally* that for undirected graphs with a multiplicative stretch better than 3 must take $\Omega(n^2)$ bits of space. This of course also applies in the presence of failures and is the reason why all the above f-DSOs have at least quadratic space. Like for distance oracles, there has been a line of work focusing on the design of f-DSOs with subquadratic space, sidestepping the bound. These oracles must have stretch (α, β) with $\alpha + \beta \geqslant 3$ since a stretch (α, β) can also be stated as $(\alpha + \beta, 0)$. Chechik, Langberg, Peleg, and Roditty [CLPR12] designed an f-DSO that, for any integer parameter $k \geqslant 1$, has space of $O(fkn^{1+1/k} \log(nW))$, query time $\widetilde{O}(|F| \log \log d_{G-F}(s,t))$, and guarantees a multiplicative stretch of $(8k-2)(f+1)$. Note that the stretch depends on the sensitivity parameter f. Recently, two f-DSOs with subquadratic space and stretch independent from f have been developed. The first one is by Bilò, Choudhary, Cohen, Friedrich, Krogmann, and Schirneck [BCC+23] that can handle up to $f = o(\log n / \log \log n)$ edge failures and, for every integer $k \geqslant 2$, guarantees a stretch of $2k-1$. The size and the query time depend on some trade-off parameter $\alpha \in (0, 1/k)$. They are equal to $kn^{1+\alpha+\frac{1}{k}+o(1)}$ and $\widetilde{O}(n^{1+\frac{1}{k}-\frac{\alpha}{k(f+1)}})$, respectively. The second f-DSO by Bilò et al. [BCC+24] works for unweighted graphs G with unique shortest paths. For every constants $f \geqslant 2$, $0 < \gamma < 1/2$, and any $\varepsilon > 0$ (possibly depending on m, n, and f), their oracle has stretch $3+\varepsilon$, space $\widetilde{O}(n^{2-\frac{\gamma}{f+1}}) \cdot O(\log n/\varepsilon)^{f+2}$, query time $O(n^\gamma/\varepsilon^2)$, and preprocessing time $\widetilde{O}(mn^{2-\frac{\gamma}{f+1}}) \cdot O(\log n/\varepsilon)^{f+1}$.

1.4 Organization of the Paper

The next section provides an overview of the techniques we use to prove our main theorems. In Section 3, we recall the central definitions. Section 4 treats the distance oracles. This includes the $(1+\varepsilon, 2W)$-stretch DO, as well as a hierarchy of DOs that trade a smaller size for a higher stretch. It follows a two-step black-box reduction turning a (static) DO into a fault-tolerant distance sensitivity oracle. The first step, in Section 5, constructs an f-DSO$^{\leqslant L}$ for hop-short distances as an intermediate data structure. Then, in Section 6, the general f-DSOs is computed. The results in Sections 4 and 6 are initially phrased as random algorithms, Section 7 derandomizes them. The paper is concluded in Section 8, where we combine the DO and the reduction to prove Theorem 5.

2 Technical Overview

We now give a more detailed overview how we achieve the results stated above. The sections corresponds to the two main parts of the paper. The first is about our new construction of distance oracles. The second part describes the framework that turns (static) DOs into distance *sensitivity* oracles. This consists of two steps: first obtain an f-DSO$^{\leqslant L}$ and then use if for the general f-DSO. In the third part of the overview, we briefly sketch our derandomization approach.

2.1 Improved Distance Oracles

Our distance oracles introduce a small additive stretch in order to make the multiplicative stretch arbitrarily small while, at the same time, keep the space subquadratic. We assume the input graph to be undirected, for it is known that subquadratic-space DOs are impossible for digraphs [TZ05]. The edges may have non-negative weights from a domain of polynomial size[3] with maximum weight W. A common pattern in the design of distance oracles is to designate a subset (or hierarchy of subsets) of vertices called *centers* [TZ05], *landmark vertices* [AG13] or *pivots* [Che14, Che15]. The data structure stores, for each vertex v in the graph, the distance from v to all pivots. Also, v knows its closest pivot $p(v)$. When given two query vertices $s, t \in V$, the oracle first checks whether s and t are sufficiently "close" to work out the exact distance $d(s,t)$. The definition of "close" varies among the different constructions in the literature. If s and t are instead "far" from each other, the estimate $d(s,p(s)) + d(p(s),t)$ is returned, which is at most $d(s,t) + 2\,d(s,p(s))$. Since s and t are "far" compared to $d(s,p(s))$, the estimate has a good stretch. This observation, of course, is in no way confined to the vertex s. For any vertex v on a shortest s-t-path, $d(s,p(v)) + d(p(v),t)$ incurs an error of at most $2\,d(v,p(v))$. This gives some freedom on how much storage space and query time one is willing to spend on finding such a v with a small distance to its closest pivot.

Our twist to that method is to look at the vicinity of a vertex not in terms of a fixed radius, but by an absolute bound on the number of considered vertices. Namely, we define a cut-off value K and store, for each vertex v, the K nearest vertices, regardless of the actual distance to v. Choosing $\widetilde{O}(n/K)$ pivots ensures that every vertex has a pivot in its K-vicinity. Storing the distances from every vertex to every pivot takes $\widetilde{O}(n^2/K)$ space, so K is our saving over quadratic space.

One of two things can happen when searching along the shortest path from s to t for a suitable vertex v. If all vertices in the list $K[v]$ have a small graph distance to v, also the closest pivot $p(v)$ must be nearby. Otherwise, there are elements in $K[v]$ that have a large graph distance, which we can use to skip ahead in the path. This sets up a win-win strategy. Consider the auxiliary graph H on the same vertex set as G in which any vertex v has an edge to each member of $K[v]$. Given a query (s,t) and an approximation parameter $\varepsilon > 0$, we conduct a bidirectional breath-first search in H starting from both s and t, trimming the search at hop-distance $1/\varepsilon$. The two searches meeting in one or more vertices is our definition of s and t being "close". We can then compute $d(s,t)$ exactly by minimizing $d(s,v) + d(v,t)$ over the intersecting vertices. Otherwise, we prove that the reason why the searches remained disjoint was that we could not skip ahead fast enough. There must have been a vertex v on the shortest s-t-path for which all neighbors in $K[v]$ have a small distance to v, including $p(v)$. We take "small" to mean at most $\frac{\varepsilon}{2}\,d(s,t) + W$, where W is the maximum edge weight. The sum $d(s,p(v)) + d(p(v),t)$ thus overestimates the true distance by at most $2\,d(v,p(v))$, resulting in an $1+\varepsilon$ multiplicative stretch and $2W$ additive.

Spacewise, the bottleneck is to store all the distances between vertices and pivots. In a second construction, we devise a way to further reduce the space, at the cost of increasing both the multiplicative and additive stretch. We are now looking for two vertices u and v on the s-t-path that both have small distance to their respective pivots $p(u)$ and $p(v)$. The portion of the distance between $p(u)$ and $p(v)$ is not stored directly but instead estimated at query time by another, internal, distance oracle. Since the inner data structure only needs to answer queries between pivots, we can get a $2k-1$ multiplicative stretch (for this part) with only $\widetilde{O}((n/K)^{1+1/k})$ space. This results in a hierarchy of new DOs with ever smaller space.

[3] The domain size is such that any graph distance can be encoded in a constant number of machine words.

2.2 From Hop-Short Distance Oracles to Distance Sensitivity Oracles

An f-DSO is a data structure that receives a query (s, t, F), consisting of vertices s and t as well as a set $F \subseteq E$ of at most f edges, and answers with an approximation of the s-t-distance $d(s, t, F)$ in the graph $G-F$. Let L be an integer cut-off parameter. A query (s, t, F) is *hop-short*, if s and t are joined by a path on at most L edges in $G-F$. An f-DSO for hop-short distances (f-DSO$^{\leqslant L}$) only guarantees a good stretch for hop-short queries. Such oracles are used as stepping stones towards general f-DSOs [ACC19, KP21, Ren22, WY13].

Recently, Bilò et al. [BCC+24] presented a new approximate f-DSO for unweighted graphs with unique shortest paths taking only subquadratic space. Implicit in their work is a pathway that takes the (static) distance oracle of Thorup and Zwick with multiplicative stretch 3 and makes it fault-tolerant increasing the stretch to $3 + \varepsilon$. The parameter $\varepsilon > 0$ influences the space, query time, and preprocessing time of the data structure. The transformation has two major steps. In the first one, the DO is used to build a hop-short f-DSO$^{\leqslant L}$. The second step then combines the answer for hop-short paths into good approximations for arbitrary queries. Their ad-hoc method is highly tailored towards the DO of Thorup and Zwick [TZ05] and does not readily generalize.

We take the same two-step approach but give an entirely new construction for the hop-short f-DSO$^{\leqslant L}$. This is necessary to make it compatible with other distance oracles. Our stand-alone framework works with any path-reporting DO as a black box. The input oracle can have an arbitrary multiplicative stretch α and may even have an additive component β, requiring new techniques. The resulting distance sensitivity oracle has stretch $(\alpha(1+\varepsilon), \beta)$. The key feature of the reduction is that, as long as the input DO has subquadratic space, so does the f-DSO.

Fault tolerance for hop-short paths. Naively, an oracle with sensitivity f must be able to handle $O(n^2 m^f)$ queries, one for each pair of vertices and any set F of up to f edge failures. Clearly, not every failure set is relevant for every pair of vertices. As a first key tool we use *fault-tolerant trees* (FT-trees) to zero in on the relevant queries. The were originally introduced by Chechik et al. [CCFK17]. We combine it with hop-short paths. There is a tree $FT(s, t)$ for every pair of vertices s and t whose shortest path in the original graph G has at most L edges. In the root that path is stored. For any edge e_1 along that path, the root has a child node which in turn stores a shortest s-t-path in the graph $G - \{e_1\}$. This corresponds to failing e_1 and looking for a *replacement path* $P(s, t, \{e_1\})$. The construction is iterated for each child node until depth f is reached. That means, for any node x at level k of $FT(s, t)$, let $F_x = \{e_1, \ldots, e_k\}$ be the edges associated with the path the root to x. The node x stores a shortest s-t-path P_x in $G-F_x$. If the shortest s-t-replacement path has more than L edges, x is made a leaf, marked by setting $P_x = \perp$.

To handle a query (s, t, F), in any node x starting with the root, it is checked whether $E(P_x) \cap F \neq \emptyset$. If so, the search continues with the first child node associated with an edge in $E(P_x) \cap F$. Otherwise, there are two cases. Either x stores some shortest s-t-path disjoint from F or $P_x = \perp$. In the first case, the length $|P_x|$ is the desired *replacement distance* $d(s, t, F)$. In the second, it is enough to report that $d(s, t, F) > L$. This reduces the number of relevant queries to $O(n^2 L^f)$.

The second key tool is an (L, f)-*replacement path covering* (RPC) [WY13, KP21]. This is a family of $\mathcal{G}$ subgraphs of G such that for any set F of at most f edges and pair of vertices s and t for which there is a replacement path $P(s, t, F)$ of at most L edges, there exists a subgraph $G_i \in \mathcal{G}$ such that $E(G_i) \cap F = \emptyset$ and G_i contains $P(s, t, F)$. RPCs are common in the design of f-DSO$^{\leqslant L}$ because, if one can find G_i quickly, the replacement distance is just $d_{G_i}(s, t)$. Recently, Karthik and Parter [KP21] gave a construction with $O(fL \log n)^{f+1}$ graphs. We shave an $(fL \log n)$-factor from that. In the full version of [KP21], they give an algorithm that takes a list $(F_1, P_1), \ldots (F_\ell, P_\ell)$ of pairs of edge sets, with $|F_k| \leqslant f$ and $|P_k| \leqslant L$. It computes a family of subgraphs that only for pairs (F_k, P_k) from the list guarantees some G_i avoiding F_k but containing P_k. We use nodes of the

fault-tolerant trees to compute a list that allows us to cover all relevant queries but get a smaller family with only $O(fL \log n)^f$ graphs. This improvement may be of independent interest.

Unfortunately, in the subquadratic-space regime, both the FT-trees as well as the graphs of the replacement-path covering are too large. The former have size $O(n^2 L^{f+1})$ (each node stores up to L edges), and the latter $O(m) \cdot O(fL \log n)^f$. Instead, we build a data structure that *emulates* queries in the FT-trees without actually storing them. The graphs in the RPC are replaced by a distance oracles. When using an (α, β)-approximate DO, this gives a saving over quadratic space. The main difficulty is that the emulated query procedure must follow the exact same path as in the actual (discarded) tree $FT(s,t)$ in order to find the correct distance oracle.

From hop-short to general distance sensitivity oracles. The second step of the transformation takes the f-DSO$^{\leqslant L}$ for hop-short distances and combines it with techniques to stitch together the answer for hop-long paths from those for hop-short ones. This step follows the pathway in [BCC+24] more closely. The key differences are the introduction of what we call *pivot trees* as well as a more thorough analysis that is needed to deal with additive stretch.

We reuse the idea of FT-trees but, as before, they are too large to be stored. Above, we emulated a query without access to the actual trees. Here, we do store some of the trees but in smaller versions and not for all pairs of vertices. Since we must also handle hop-long queries, any node of an FT-tree may now store a path of up to n edges. If we were to create a child node for each edge, we would end up with a prohibitive space of $O(n^{f+1})$ for a single tree. Instead, we use larger segments and each child node now corresponds to the failure of a whole segment. Of course, this has the danger that failing a segment may inadvertently destroy replacement paths that would still exists in $G - F$, where only the failing edges are removed. We have to strike a balance between the space reduction of larger segments and the introduced inaccuracies. This leads to the concept of granularity. In an FT-tree with granularity λ, the first and last $\lambda/2$ edges form their own segments. Beyond that, we use segments whose size increase exponentially towards the middle of the stored path. The base of this exponential is $1 + \Theta(\varepsilon)$, where $\varepsilon > 0$ is the approximation parameter. A higher granularity means much larger trees but with higher accuracy.

We offset this by building the larger trees only for very few pairs of vertices. We sample a set B_1 of $\widetilde{O}(fn/L^f)$ pivots and build an FT-tree with granularity $\lambda = \Theta(\varepsilon L)$ for each pair of vertices in $B_1 \times B_1$. For some vertex v and failure set F, let $\text{ball}_{G-F}(v, \lambda/2)$ be the ball of hop-distance $\lambda/2$ around v in $G-F$. We show that if both s and t have respective pivots $p_s \in \text{ball}_{G-F}(s, \lambda/2) \cap B_1$ and $p_t \in \text{ball}_{G-F}(t, \lambda/2) \cap B_1$, then the FT-tree $FT_\lambda(p_s, p_t)$ with granularity λ gives a very good estimate of the replacement distance $d(s, t, F)$. However, since there are so few pivots in B_1 this can only be guaranteed if the balls around s and t are very dense. We also have to give a fall-back solution in case one of the balls is sparse. We sample a set B_2 of now $\widetilde{O}(fn/L)$ pivots (much more than B_1) and build an FT-tree *without* granularity for pairs of vertices in $B_2 \times V$. These trees are much smaller and we show that together with the f-DSO$^{\leqslant L}$ they are still sufficiently accurate.

The problem is to quickly find the pivots that are close to s and t in $G-F$ at query time. Simply scanning over B_1 and B_2 is way too slow. Instead, we use yet another kind of tree data structure, *pivot trees*. They are inspired by FT-trees but are not the same. We have one pivot tree per vertex in V (instead of an FT-tree for every pair). The tree belonging to s stores in each node x some shortest path P_x starting from s as before, and each child of node x represents the failure of one edge of P_x. The key difference is that the other endpoint of P_x can vary now, the path ends in the closest pivot $p_s \in B_1$ of the first type. Note that in the graph $G-F_x$ that pivot may not be the same as in $G-(F_x \cup \{e_{k+1}\})$. We only care about the vicinity $\text{ball}_{G-F}(s, \lambda/2)$, so the paths P_x have at most $\lambda/2$ edges. If the closest pivot in B_1 is too far away, $\text{ball}_{G-F}(s, \lambda/2)$ must be sparse. That means, also $\text{ball}_{G-F}(s, \lambda/2) \cap B_2$ is small and it is feasible to store all pivots of the second type.

Unfortunately, this is still not enough. Due to our sparsification via segments with multiple edges, the answer of the FT-trees with and with out granularity are only accurate if the replacement path $P(s,t,F)$ is "far away" from all failures in F. That roughly means that any vertex of $P(s,t,F)$ is more than an $\Theta(\varepsilon)$-multiple from any endpoint of some edge in F. If this safety area is free of failures, no segment can accidentally contain an edge of F, which would disturb the return value of the FT-tree too much. If, however, there is a failure too close to the path, Chechik et al. [CCFK17] showed that there is always some surrogate target $t' \in V(F)$ such that the replacement path $P(s,t',F)$ is indeed far away from all failures and also not much of a detour compared to going directly from s to t. This causes an $1+\varepsilon_1$ factor in the multiplicative part of the stretch, where $\varepsilon_1 = \Theta(\varepsilon)$. We build an auxiliary *weighted* complete graph H on the vertex set $V(F) \cup \{s,t\}$ to exploit this detour structure. We use the FT-trees and f-DSO$^{\leqslant L}$ to compute the weight of all edges in the graph. The eventual answer of our oracle is the s-t-distance in H.

The main obstacle is to prove that $d_H(s,t)$ is actually an $(\alpha(1+\varepsilon),\beta)$-approximation of $d(s,t,F)$. This is also the other decisive difference to [BCC+24] in that we need to handle both the multiplicative and the additive stretch. Consider an edge $\{u,v\} \in E(H)$. If $P(u,v,F)$ is hop-short, the f-DSO$^{\leqslant L}$ trivially gives an (α,β)-approximation of $d(u,v,F)$ which we use to compute the weight $w_H(u,v)$. If $P(u,v,F)$ has more than L edges and is "far away" from all failures, we show that $w_H(u,v)$ computed by the FT-trees is only an $(\alpha, X\beta)$-approximation. The blow-up $X = O(f\log(n)/\varepsilon)$ stems from the segments of increasing size. This is only exacerbated by the fact that the shortest s-t-path in the weighted graph H has $O(f^2)$ edges, each one contributing their own distortion to the additive stretch. However, the *additive* blow-up only happens if $P(u,v,F)$ has many edges, that is, if $d(u,v,F)$ is large. The idea is to then charge most of the additive stretch to the multiplicative part, increasing it only by another $1+\varepsilon_2 = 1+\Theta(\varepsilon)$ factor which is then combined with the $1+\varepsilon_1$ stemming from the auxiliary graph H. To make this work, we carefully analyze the interplay of the edges in H, using an induction over $E(H)$ in the order of increasing replacement distance $d(u,v,F)$.

2.3 Derandomization

For the derandomization, we use the framework of critical paths proposed by Alon, Chechik, and Cohen [ACC19]. It consists of identifying a small set of shortest paths in G, (greedily) compute a deterministic hitting set for them, and then build the fault-tolerant data structure from there. The number of paths needs to be small in order to keep the derandomization efficient. This is the main difficulty in the approach. Our threshold of efficiency is that making the data structures deterministic should not increase their preprocessing time by more than a constant factor.

In the context of derandomization, we view paths as mere sets of vertices without any further structure. This allows us to apply the approach also to other subsets of V in a unified fashion. Consider the (static) distance oracle for weighted graphs in Theorems 1 and 3. The list $K[v]$, containing the K vertices closest to v, are key components in the construction. It is indeed enough to hit all the $\{K[v]\}_{v\in V}$ to derandomize the DO. This also incurs hardly any extra work as those lists are compiled anyway in the preprocessing.

The construction of the general f-DSO from the one restricted to hop-short distances (Theorem 7) builds on the pivot sets B_1 and B_2 whose derandomization is more involved. Recall that we use λ for the granularity of the FT-trees. During the proof of correctness, it becomes apparent that the pivots of the second type in B_2 need to hit the length-$\lambda/2$ prefixes and suffixes of all concatenations of up to two replacement paths, provided that those concatenations are hop-long (have more than L edges). We use a structural result by Afek, Bremler-Barr, Kaplan, Cohen, and Merritt [ABK+02] that states that replacement paths themselves are concatenations of $O(f)$ short-

est paths in the original graph G. This allows us to show that computing a deterministic hitting set of all shortest paths in G of length $\Omega(\lambda/f)$ suffices to guarantee the same covering properties as B_2. The set B_1, in turn, need to hit all the sets $\text{ball}_{G-F}(u, \lambda/2)$ that are sufficiently dense (more than L^f vertices). In principle, a similar approach as for B_2 would work but that would either produce way too many pivots or take much to long. Instead, we interleave the level-wise construction of the pivot trees with derandomization phases to find a deterministic stand-in for B_1.

3 Preliminaries

We let $G = (V, E)$ denote the base graph with n vertices and m edges. We tacitly assume $m \geqslant n$. Depending on the setting G is either directed or undirected, may be edge-weighted by non-negative reals or unweighted, as stated in the respective sections. In case G is weighted and undirected, we assume that the weight function w maps into an subset of the interval $[1, W]$ of at most m elements. This does not lose generality as we can contract (undirected) weight-zero edges and scaling all other weights by $1/\min_{e\in E} w(e)$. Note that any distance in the graph can be encoded in a constant number of machine words on $O(\log n)$ bits. This is the reason why none of the asymptotic bounds depend on W.

For a 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 $P = (u_1, \ldots, u_i)$ and $Q = (v_1, \ldots, v_j)$ be two paths in H. Their *concatenation* is $P \circ Q = (u_1, \ldots, u_i, v_1, \ldots, v_j)$, which is well-defined if $u_i = v_1$ or $\{u_i, v_1\} \in E(H)$. For vertices $u, v \in V(P)$, we let $P[u..v]$ denote the subpath of P from u to v. The *length* of path P is $w(P) = \sum_{e\in E(P)} w(e)$. We use $|P| = |E(P)|$ for the number of edges. For $s, t \in V(H)$, the *distance* $d_H(s, t)$ is the minimum length of any s-t-path in H; and $d_H(s, t) = +\infty$ if no such path exists. We drop the subscripts for the input graph G.

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.) Let α, β be two non-negative real numbers with $\alpha \geqslant 1$. A *distance oracle* (DO) for H is a data structure that reports, upon query (s, t), an approximation of the distance $d_H(s, t)$. It has *stretch* (α, β) if the reported value $\widehat{d}_H(s, t)$ satisfies $d_H(s, t) \leqslant \widehat{d}_H(s, t) \leqslant \alpha \cdot d_H(s, t) + \beta$. We say the stretch is *multiplicative* if $\beta = 0$, *additive* if $\alpha = 1$ and *near-additive* if $\alpha = 1+\varepsilon$ for some parameter $\varepsilon > 0$ that can be made arbitrarily small.

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 $d(s, t, F) = d_{G-F}(s, t)$ is the *replacement distance*. For a positive integer f, an *f-edge fault-tolerant distance sensitivity oracle* (f-DSO) answers queries (s, t, F), with $|F| \leqslant f$, with an approximation of the replacement distance $d(s, t, F)$. The stretch of an f-DSO is defined as for DOs. The maximum number f of supported failures is the *sensitivity*. Let $L \leqslant n$ be a positive integer parameter. We also discuss *f-DSO for hop-short distances* bounded by L (f-DSO$^{\leqslant L}$). On query (s, t, F), their estimate $\widehat{d^{\leqslant L}}(s, t, F)$ must always satisfy $\widehat{d^{\leqslant L}}(s, t, F) \geqslant d(s, t, F)$. The upper bound of the stretch requirement $\widehat{d^{\leqslant L}}(s, t, F) \leqslant \alpha \cdot d(s, t, F) + \beta$ only needs if there is a shortest path from s to t in $G-F$ with at most L edges.

We measure the space complexity of distance (sensitivity) oracles and all other data structures in the number of $O(\log n)$-bit words. The size of the graph G does not count against the space, unless it is stored explicitly.

Algorithm 1: Query algorithm of the DO in Theorem 1 for the query (s,t). d_H^{hop} is the hop-distance in H, $d_{1/\varepsilon}$ is the minimum length of all paths with at most $\lceil 1/\varepsilon \rceil$ hops in H, and $p(v) \in B$ is the pivot closest to v in G.

$A_{1/\varepsilon}(s) \leftarrow \{v \in V \mid d_H^{\text{hop}}(s,v) \leqslant \lceil 1/\varepsilon \rceil\}$;
$A_{1/\varepsilon}(t) \leftarrow \{v \in V \mid d_H^{\text{hop}}(v,t) \leqslant \lceil 1/\varepsilon \rceil\}$;
$\widehat{d}_1 \leftarrow \min\{d_{1/\varepsilon}(s,v) + d_{1/\varepsilon}(v,t) \mid v \in A_{1/\varepsilon}(s) \cap A_{1/\varepsilon}(t)\}$;
$\widehat{d}_2 \leftarrow \min\{d(s,p(v)) + d(p(v),t) \mid v \in A_{1/\varepsilon}(s) \cup A_{1/\varepsilon}(t)\}$;
return $\min(\widehat{d}_1, \widehat{d}_2)$;

4 Distance Oracles for Static Graphs

We now present the construction of our (static) distance oracles in subquadratic space, trading a small additive stretch for an improved multiplicative one. We first discuss a near-additive oracle with a multiplicative stretch $1+\varepsilon$ and additive stretch $2W$. Upon request, it also reports the approximate shortest path, which we use in Section 5. It has a space of $\widetilde{O}(n^2/K)$ for some integer parameter K. We then show how to reduce the space to $\widetilde{O}((n/K)^{1+1/k})$ at the expense of raising the multiplicative stretch to $2k-1+\varepsilon$, and the additive to $4kW$.

4.1 Near-Additive Distance Oracle

For convenience, we restate Theorem 1 below. In this section we prove a randomized version in which the features of the oracle are only guaranteed with high probability. The derandomization is deferred to Section 7.1. We first treat the space, query time, and stretch of the DO. Subsequently, we describe how to make the oracle path-reporting.

Theorem 1. *Let W be a non-negative real number, and G an undirected graph with n vertices and edge weights in a $\mathsf{poly}(n)$-sized subset of $[0,W]$. For every positive integer $K \leqslant \sqrt{n}$ and any $\varepsilon > 0$, there exists a path-reporting distance oracle for G that has stretch $(1+\varepsilon, 2W)$, space $\widetilde{O}(n^2/K)$, and query time $O(K^{\lceil 1/\varepsilon \rceil})$ for the distance and an additional $O(1)$ time per reported edge. The data structure can be constructed in APSP time.*

Preprocessing and space. Let G denote the underlying graph. First, we make a simplifying assumption. It is easy to compute the connected components of G in time $O(m)$ (a fortiori in APSP time) and store an $O(n)$-sized table of the component IDs for all vertices. This allows us to check in constant time whether two vertices have a path between them, and otherwise correctly answer $d(s,t) = +\infty$. We can thus assume to only receive queries for vertex pairs in the same component.

In APSP time, we compute, for every vertex $v \in V$, the list $K[v]$ of its K closest vertices in G (including v itself) where ties are broken arbitrarily. Each element of $K[v]$ is annotated with its distance to v. We also sample a set B of vertices, called *pivots*, by including any vertex in B independently with probability $C(\log n)/K$ for a sufficiently large constant $C > 0$. It is easy to show using Chernoff bounds that with high probability B has $O(\frac{n}{K} \log n)$ elements. We use $p(v)$ to denote the pivot closest to v. Our data structure stores, for each vertex v, the list $K[v]$, the closest pivot $p(v)$, and the distance $d(v,p(v))$. Moreover, for each pivot $p \in B$, it additionally stores the distance from p to *every* vertex in G. The data structure takes space $O(|B|n + nK) = O(\frac{n^2}{K} \log n + nK)$,

which is $\widetilde{O}(n^2/K)$ for $K = O(\sqrt{n})$.

Query algorithm. We use an auxiliary graph H to simplify the presentation of the query algorithm and the subsequent reasoning. The graph H has the same vertex set as G and, for each $v \in V$, an edge from v to any $v' \in K[v]\backslash\{v\}$ whose weight is $d(v, v')$. The *hop-distance* between two vertices $u, v \in V$ in H, is the minimum number of edges on any u-v-path.

Algorithm 1 summarizes how a query $(s, t) \in V^2$ is processed. Let $A_{1/\varepsilon}(s), A_{1/\varepsilon}(t)$ be the sets of vertices with hop-distance at most $\lceil 1/\varepsilon \rceil$ from s and t, respectively, in the graph H. To compute the set $A_{1/\varepsilon}(s)$ we use a slightly modified breath-first search from s (analogously for $A_{1/\varepsilon}(t)$ starting from t). The modification consists in exploring, in each step with current vertex v, *all* neighbors in $K[v]\backslash\{v\}$ as long as fewer than $\lceil 1/\varepsilon \rceil$ hops have been made. In particular, the search revisit vertices that have been encountered earlier. Each time a vertex v is visited, its estimate of the distance $d_H(s, v)$ is updated to the minimum over all paths explored so far. In other words, for each $v \in A_{1/\varepsilon}(s)$ the length of the shortest path from s that uses at most $\lceil 1/\varepsilon \rceil$ edges is computed. We use $d_{1/\varepsilon}(s, v)$ to denote this distance. Note that $d_{1/\varepsilon}(s, v)$ may overestimate the true distance $d_H(s, v)$ in H, which in turn overestimates $d(s, v)$ in G. Below, we identify some conditions under which the estimate is accurate. The sets $A_{1/\varepsilon}(s), A_{1/\varepsilon}(t)$ have $O(K^{\lceil 1/\varepsilon \rceil})$ elements and, with the information stored by the DO, the modified BFSs in H can be emulated in time $O(K^{\lceil 1/\varepsilon \rceil})$.

Recall that $p(v)$ is the pivot closest to v and that the values $d(s, p(v)), d(p(v), t)$ are stored. The oracle uses $A_{1/\varepsilon}(s), A_{1/\varepsilon}(t)$ to compute

$$\widehat{d_1} = \min_{v \in A_{1/\varepsilon}(s) \cap A_{1/\varepsilon}(t)} d_{1/\varepsilon}(s, v) + d_{1/\varepsilon}(v, t) \quad \text{and} \quad \widehat{d_2} = \min_{v \in A_{1/\varepsilon}(s) \cup A_{1/\varepsilon}(t)} d(s, p(v)) + d(p(v), t).$$

It returns the smaller of the two estimates. The total query time is $O(K^{\lceil 1/\varepsilon \rceil})$.

Stretch. For a radius r and vertex $v \in V$, let $\text{ball}(v, r)$ be the set of all vertices that have distance at most r from v. Define $V_r = \{v \in V \mid \text{ball}(v, r) \subseteq K[v]\}$, that is, the set of all vertices v that have at most K vertices within distance r. Recall that for any query (s, t) to the DO, we can assume that there is a path between s and t in G. In the following lemma, the additive term W can be replaced by the largest weight w of an edge on the path P. This would reduce the stretch in Lemma 9 to $(1+\varepsilon, 2w)$.

Lemma 8. *Consider two vertices $s, t \in V$ with distance $d(s, t)$ and set $r = \frac{\varepsilon}{2} d(s, t) + W$. Let P be a shortest path between s and t in G.*

(i) If all vertices of P lie in V_r the hop-distance between s and t in H is at most $\lceil 2/\varepsilon \rceil$.

(ii) If all vertices of P lie in V_r, then $V(P) \cap A_{1/\varepsilon}(s) \cap A_{1/\varepsilon}(t)$ is non-empty.

(iii) If P contains a vertex from $V\backslash V_r$, then $(V(P) \setminus V_r) \cap A_{1/\varepsilon}(s)$ or $(V(P) \setminus V_r) \cap A_{1/\varepsilon}(t)$ is non-empty.

Proof. Let d abbreviate the distance $d(s, t)$. First, consider the case that all vertices of P lie in $V_r = V_{\frac{\varepsilon d}{2} + W}$. We define a sequence σ_s of vertices in $V(P)$, namely, a subsequence of P when directed away from s. Its first vertex is $x_0 = s$; each consecutive vertex x_{i+1} is the element of $V(P[x_i, t]) \cap \text{ball}(x_i, r)$ that maximizes $d(x_i, x_{i+1})$. That means x_{i+1} comes after x_i on P when going from s to t, but it has distance at most $\frac{\varepsilon d}{2} + W$ from x_i. By the assumption $V(P) \subseteq V_r = \{v \in V \mid \text{ball}(v, r) \subseteq K[v]\}$, the sequence σ_s corresponds to an actual sequence of hops in the auxiliary graph H. Moreover, for each x_i, we have $d_{1/\varepsilon}(s, x_i) \leqslant \sum_{j=0}^{i-1} d(x_j, x_{j+1}) = d(s, x_i)$. So in this case the estimate $d_{1/\varepsilon}(s, x_i)$ is exact.

Observe that consecutive vertices in σ_s (except possibly the last pair involving t) have graph distance in G of more than $\varepsilon d/2$. Indeed, if we had $d(x_i, x_{i+1}) \leqslant \varepsilon d/2$, then the next vertex v on P that comes after x_i has a higher distance, but still satisfies $d(x_i, v) = d(x_i, x_{i+1}) + w(x_{i+1}, v) \leqslant \frac{\varepsilon d}{2} + W$. This shows that σ_s reaches from s to t in at most $\lceil 2/\varepsilon \rceil$ hops.

Symmetrically, we define the sequence σ_t by perceiving P as directed away from t. The first vertex is $y_0 = t$ and y_{i+1} maximizes $d(y_i, y_{i+1})$ over $V(P[s, y_i]) \cap \mathrm{ball}(y_i, r)$. Let k be the smallest index in σ_s such that $d(s, x_k) \geqslant d/2$. Then, we have $k \leqslant \lceil 1/\varepsilon \rceil$ by the above observation on the consecutive distances. Analogously, let ℓ be the minimum index in σ_t with $d(t, y_\ell) \geqslant d/2$. The predecessor of y_ℓ thus satisfies $\frac{d}{2} < d(s, y_{\ell-1}) \leqslant \frac{d}{2} + r$ and therefore $d(x_k, y_{\ell-1}) \leqslant r$. This shows that x_k can be reached from t in H via $\ell \leqslant \lceil 1/\varepsilon \rceil$ hops by first following σ_t until $y_{\ell-1}$ and then hopping to x_k. In particular, we have $x_k \in V(P) \cap A_{1/\varepsilon}(s) \cap A_{1/\varepsilon}(t)$.

The last part, where the path P contains a vertex from $V \setminus V_r$, is structurally similar but somewhat simpler. Let vertex z be the minimizer of $\min(d(s, z), d(z, t))$ in $V(P) \setminus V_r$. W.l.o.g. z is closer to s than to t, whence $d(s, z) \leqslant d/2$. We now define the sequence σ_s as above but let it end in z instead of t. Note that, by the minimality of z, all vertices of σ_s except for z itself are in V_r. The same argument as above now shows that σ_s has at most $\lceil 1/\varepsilon \rceil$ vertices, which correspond to hops in H. That means, $z \in (V(P) \setminus V_r) \cap A_{1/\varepsilon}(s)$. □

To prove the approximation guarantee, we use the following straightforward application of Chernoff bounds: with high probability all vertices $v \in V \setminus V_r$ satisfy $d(v, p(v)) \leqslant r$. For if $\mathrm{ball}(v, r)$ contains more than K elements, it also has a pivot w.h.p.

Lemma 9. *With high probability over all queries $(s, t) \in V^2$, the distance oracle returns an estimate of $d(s, t)$ with stretch $(1 + \varepsilon, 2W)$.*

Proof. The oracle correctly answers $+\infty$ if s and t are in different component. Let again P be a shortest s-t-path, $d = d(s, t)$, and $r = \frac{\varepsilon d}{2} + W$. First, note that the returned value is never smaller than d since $\widehat{d_1} = d_{1/\varepsilon}(s, v) + d_{1/\varepsilon}(v, t) \geqslant d(s, v) + d(v, t) \geqslant d(s, t)$ and $\widehat{d_2} = d(s, p(v)) + d(p(v), t)$ even corresponds to an actual path between s and t.

First, assume that all vertices of P are in V_r. There exists some vertex $v \in V(P) \cap A_{1/\varepsilon}(s) \cap A_{1/\varepsilon}(t)$. The returned distance is exact since

$$\widehat{d_1} \leqslant d_{1/\varepsilon}(s, v) + d_{1/\varepsilon}(v, t) = d(s, v) + d(v, t) = d.$$

The first equality was argued in Lemma 8, the second one is due to v being on a shortest s-t-path.

If $V(P) \not\subseteq V_r$, there exists some $v \in (V(P) \setminus V_r) \cap (A_{1/\varepsilon}(s) \cup A_{1/\varepsilon}(t))$, from which we get w.h.p.

$$\widehat{d}(s, t) \leqslant \widehat{d_2} \leqslant d(s, p(v)) + d(p(v), t) \leqslant d(s, v) + d(v, t) + 2 \cdot d(v, p(v)) \leqslant d + 2r = (1+\varepsilon)\, d + 2W. \quad \square$$

Reporting paths. Only small adaptions are needed to make the oracle path-reporting. Recall that we use an emulated BFS to compute the distance $d_{1/\varepsilon}(s, v)$ for all $v \in A_{1/\varepsilon}(s)$ (length $d_{1/\varepsilon}(v, t)$ for $v \in A_{1/\varepsilon}(t)$) by iteratively updating the current best length of a path in H with at most $\lceil 1/\varepsilon \rceil$ edges. A path through H may not correspond to a path through G, namely, if it uses and edge $\{v, v'\}$ with $v' \in K[v]$ that is not actually present in G. However, any shortest v-v'-path in G exclusively uses vertices from $K[v]$.

We thus store $K[v]$ in the form of a shortest-path tree in G rooted at v that is truncated after K vertices have been reached. In each step of the emulated BFS, we explore the neighborhood $K[v]$ in the order given by an actual BFS of the shortest-path tree. Each time some estimate $d_{1/\varepsilon}(s, v')$ is updated we also store a pointer to the last vertex from which we reached v'. Furthermore, with

each distance $d(v,p)$ for an arbitrary pivot $p \in B$, we store the first edge on a shortest path from v to p. This does not change the space requirement of the oracle by more than a constant factor.

Suppose the minimum reported in line 5 of Algorithm 1 is attained by $\widehat{d_1}$ and let v be the minimizing vertex in line 3. Following the stored pointers backwards reconstructs a shortest s-v-path through $A_{1/\varepsilon}(s)$. Symmetrically, following the pointers forward gives shortest v-t-path through $A_{1/\varepsilon}(t)$. Reporting this path in order from s to t can be done by visiting any of the computed edges at most twice. If instead $\widehat{d_2}$ is smaller with minimizer v (in line 4), we follow the first edge on a shortest path from s to $p(v)$ (which we have stored) and likewise for each intermediate vertex we encounter this way between s and $p(v)$. After $p(v)$, we instead follow along a shortest t-$p(v)$-path.

4.2 A Hierarchy of Distance Oracles

We now discuss construction of a hierarchy of DOs for general weighted graphs. The main bottleneck of the space requirement of the DO in Theorem 1 is to store the distance from every pivot to all vertices of the graph. We next show how to improve this by instead estimating the distance between pivots with another, internal, DO. In effect, we trade ever smaller space for higher overall stretch.

Theorem 3. *Let W be a non-negative real number, and G an undirected graph with n vertices and edge weights in a $\mathsf{poly}(n)$-sized subset of $[0,W]$. For all positive integers k and K with $K = O(n^{1/(2k+1)})$, and every $\varepsilon > 0$, there exits a distance oracle for G that has stretch $(2k-1+\varepsilon, 4kW)$, space $O((\frac{n}{K})^{1+1/k} \log^{1+1/k} n)$, and query time $O(K^{2\lceil 4k/\varepsilon \rceil})$. The data structure can be constructed in APSP time.*

In order to prove this, we use the following result of Chechik [Che14, Che15]. She gave an improved implementation of the Thorup-Zwick DO [TZ05] and also obtained a leaner version of the oracle when restricting the attention to distances between only a subset of the vertices.

Theorem 10 (Chechik [Che14, Che15])**.** *Let $G = (V,E)$ be an undirected weighted graph. For any positive integer k, and vertex set $B \subseteq V$, there is a data structure that reports $(B \times B)$-distances with multiplicative stretch $2k-1$, space $O(|B|^{1+\frac{1}{k}})$, and constant query time. The preprocessing time is $O(n^2 + m\sqrt{n})$.*

Preprocessing and space. For Theorem 3, the set of pivots B is sampled from V using the probability $C \log(n)/K$. (That means, we do not apply the indirect sampling via edges here.) Let the lists $K[v]$ and closest pivot $p(v)$ be defined as in Section 4.1. We preprocess the DO of Theorem 10 for pairs of pivots in B. Let $\widehat{D}(p,q)$ denote the estimate of $d(p,q)$ returned by that DO when queried with $p,q \in B$. Our data structure stores the list $K[v]$ for every $v \in V$ as well as the $(B \times B)$-DO. This takes space $O(nK + (\frac{n}{K} \log n)^{1+1/k}) = O((\frac{n}{K})^{1+1/k} \log^{1+1/k} n)$, assuming $K = O(n^{1/(2k+1)})$.

Query algorithm. The query algorithm is shown in Algorithm 2. Recall that we defined the auxiliary graph H by requiring that every vertex is connected to its K closest vertices in G. Define $\delta = \varepsilon/(2k)$ and let $A_{2/\delta}(s), A_{2/\delta}(t)$ the sets of vertices that have a hop-distance at most $\lceil 2/\delta \rceil = \lceil 4k/\varepsilon \rceil$ from s and t, respectively, in H. If t is found while computing $A_{2/\delta}(s)$, the oracle returns the distance $d_{2/\delta}(s,t)$, that is, the minimum length of all s-t-paths in H with at most $\lceil 2/\delta \rceil$ edges. Otherwise, it reports

$$\widehat{d} = \min_{u \in A_{2/\delta}(s),\, v \in A_{2/\delta}(t)} d(s,p(u)) + \widehat{D}(p(u),p(v)) + d(p(v),t)$$

Algorithm 2: Query algorithm of the DO in Theorem 3 for the query (s,t).
d_H^{hop} is the hop-distance in H, $d_{4k/\varepsilon}$ is the minimum length of all paths with at most $\lceil 4k/\varepsilon \rceil$ hops in H, $p(v) \in B$ is the pivot closest to v in G, and $\widehat{D}$ is the output of the DO in Theorem 10.

$A_{4k/\varepsilon}(s) \leftarrow \{v \in V \mid d_H^{\text{hop}}(s,v) \leqslant \lceil 4k/\varepsilon \rceil\}$;
$A_{4k/\varepsilon}(t) \leftarrow \{v \in V \mid d_H^{\text{hop}}(v,t) \leqslant \lceil 4k/\varepsilon \rceil\}$;
if $t \in A_{4k/\varepsilon}(s)$ **then**
  return $d_{4k/\varepsilon}(s,t)$;
else
  return $\min\{d(s,p(u)) + \widehat{D}(p(u),p(v)) + d(p(v),t) \mid u \in A_{4k/\varepsilon}(s);\ v \in A_{4k/\varepsilon}(t)\}$;

Evaluating that minimum over all pairs (u,v) takes time $O(|A_{2/\delta}(s)||A_{2/\delta}(t)|) = O((K^{\lceil 2/\delta \rceil})^2) = O(K^{2\lceil 4k/\varepsilon \rceil})$, which dominates the query time.

Stretch. Fix two query vertices $s,t \in V$ and let $d = d(s,t)$ be their distance. Recall that the set $V_{\frac{\delta d}{2}+W}$ consists of all those vertices whose ball with radius $\frac{\delta d}{2} + W$ has size at most K.

Lemma 11. *With high probability over all queries $(s,t) \in V^2$, the DO returns an estimate of $d(s,t)$ with stretch $(2k{-}1{+}\varepsilon, 4kW)$.*

Proof. Let P be a shortest s-t-path in G. By Lemma 8 with δ in place of ε, if all vertices of P are in $V_{\frac{\delta d}{2}+W}$ then s and t have hop-distance at most $\lceil 2/\delta \rceil$ in the auxiliary graph H, whence $t \in A_{2/\delta}(s)$ and $d_{2/\delta}(s,t) = d(s,t)$.

Otherwise, the set $V(P) \backslash V_{\frac{\delta d}{2}+W}$ is non-empty. Let v_s be the vertex on P that is closest to s and does not lie in $V_{\frac{\delta d}{2}+W}$. Clearly, we have $v_s \in A_{2/\delta}(s)$. For the vertex $v_t \in V(P) \backslash V_{\frac{\delta d}{2}+W}$ that is closest to the other endpoint t, we get $v_t \in A_{2/\delta}(t)$. In this case, the output of our oracle is at most

$$\begin{aligned}
\widehat{d} &\leqslant d(s,p(v_s)) + \widehat{D}(p(v_s),p(v_t)) + d(p(v_t),t) \leqslant d(s,p(v_s)) + (2k{-}1)\, d(p(v_s),p(v_t)) + d(p(v_t),t) \\
&\leqslant d(s,v_s) + d(v_s,p(v_s)) + (2k{-}1) d(p(v_s),v_s) + (2k{-}1)\, d(v_s,v_t) + (2k{-}1)\, d(p(v_t),v_t) + \\
&\quad d(p(v_t),v_t) + d(v_t,t) \\
&\leqslant (2k{-}1)\, d(s,t) + 2k\, d(v_s,p(v_s)) + 2k\, d(p(v_t),v_t) \leqslant (2k{-}1)\, d + 4k \left(\frac{\delta d}{2} + W\right) \\
&= (2k{-}1{+}\varepsilon)\, d + 4kW.
\end{aligned}$$

□

5 Fault-Tolerant Tree Oracles

The aim of this section is to prove Theorem 6 (restated below), adding fault tolerance to the construction of distance oracles for short paths. The core idea is to build FT-trees during preprocessing to identify a family of relevant subgraphs of G. For those subgraphs (α,β)-stretch DOs for hop-short distances are constructed and stored. The FT-trees, however, are discarded at the end of the preprocessing as they are too large to fit in subquadratic space. The main challenge is to utilize the correct DOs at query time. For this, we *emulate* a root-to-leaf transversal in the discarded FT-tree. This motivates the name *fault-tolerant tree oracles*, abbreviated *FT-tree oracles*.

Theorem 6. *Let G be an unweighted (possibly directed) graph, with n vertices, and let f and L be positive integer parameters possibly depending on n. Assume access to a path-reporting distance oracle that, on any spanning subgraph of G, has stretch (α, β), takes space S, query time Q, and preprocessing time T. Then, there is an f-edge fault-tolerant distance sensitivity oracle for replacement paths in G with at most L edges that has*

- *stretch (α, β),*
- *space $O(fL \log n)^f \cdot \mathsf{S}$,*
- *query time $\widetilde{O}(f \cdot (\mathsf{Q} + \alpha L + \beta + f^2 L))$,*
- *preprocessing time $\widetilde{O}(n^2(\alpha L + \beta)^f) \cdot (O(fL \log n)^f + \mathsf{Q}) + O(fL \log n)^f \cdot \mathsf{T}$.*

To achieve the above theorem, we build the FT-trees along side $O(Lf \log n)^f$ path-reporting DOs with stretch (α, β) during preprocessing. When the construction of the FT-trees is completed, we dump the FT-trees and keep only the path-reporting DOs. We will show that this path-reporting DOs together with some auxiliary data structures will allow us to emulate a search in the FT-trees that were used to construct them.

The preprocessing algorithm. We now describe how the FT-trees and the path-reporting DOs are built side-by-side (see Algorithm 3 for the pseudocode). We first define the notion of (α, β)-*hit-and-miss*, which is somewhat similar to the one provided in Karthik and Parter [KP21].[4] We say that a family $\mathcal{O}$ of DOs (α, β)-hits-and-misses a node x in $FT(s,t)$, if there exists at least one DO $\mathcal{O}_x \in \mathcal{O}$ such that the path $P_{\mathcal{O}_x}(s,t)$ reported by $\mathcal{O}$ when queried with (s,t) is edge-disjoint from F, *i.e.*, $E(P_{\mathcal{O}}(s,t)) \cap F = \emptyset$ and $|P_{\mathcal{O}}(s,t)| \leqslant \alpha d(s,t,F) + \beta$. We build the FT-trees for G level by level. For each level i, we precompute a family $\mathcal{O}_i$ of path-reporting DOs for hop-short distances that hits-and-misses all the i-th level nodes x of all FT-trees. The data structure of our f-DSO is then given by the family $\{\mathcal{O}_0, \ldots, \mathcal{O}_f\}$. So, we discard all the computed FT-trees. Moreover, we compute an auxiliary data structure that, for every node x in level i of some FT-tree, can deterministically identify the *representative* DO in $\mathcal{O}_i$ that hits-and-misses x. This auxiliary data structure is necessary to simulate the search on the FT-trees whenever we need to answer to a query (s,t,F).

We now describe how we compute the FT-trees level by level.

For level 0, $\mathcal{O}_0$ contains the path-reporting DO$^{\leqslant L}$ $\mathcal{O}_G$ with stretch (α, β) that is computed using algorithm $\mathcal{A}$. For every two distinct vertices s and t of G, the root node x of $FT(s,t)$ is associated with the path P_x computed by querying the DO$^{\leqslant L}$ $\mathcal{O}_G$ with the pair (s,t). Moreover, $F_x = \emptyset$.

The generic level i, with $1 \leqslant i \leqslant f$, of the FT-trees is computed using a two stage algorithm.

In the first stage we do the following. For every two distinct vertices s and t of G, and for every node x in level i of $FT(s,t)$, we compute a path P'_x from s to t in the graph $G - F_x$ using Dijkstra's algorithm. Once the entire i-th level of all FT-trees has been computed, we compute a family of graphs $\mathcal{G}_i$ that, according to the definition provided in Karthik and Parter [KP21], *hits-and-misses* the family of pairs (P'_x, F_x), for all nodes x in level i of some FT-tree. More precisely, $\mathcal{G}_i$ hits-and-misses (P'_x, F_x) if there is $H_x \in \mathcal{G}_i$ such that H_x contains P'_x, but does not contain F_x.

In the second stage we use the algorithm $\mathcal{A}$ to build the set of DOs for hop-short distances $\mathcal{O}_i = \{\mathcal{O}_H \mid H \in \mathcal{G}_i\}$ with stretch (α, β). We then substitute all the paths associated with the nodes of the i-th level of all the FT-trees with the corresponding paths computed by querying the representative DOs for hop-short distances in $\mathcal{O}_i$. More precisely, given a node x of the i-th level of $FT(s,t)$, we pick a graph $H_x \in \mathcal{G}_i$ that hits-and-misses (P'_x, F_x), we query the DO$^{\leqslant L}$ $\mathcal{O}_{H_x}$ with

[4] For simplicity we will write hit-and-miss when (α, β) are clear from the context.

Algorithm 3: The algorithm that builds the f-DSO$^{\leqslant L}$ with stretch (α, β) of Theorem 6.

Input : An undirected graph $G = (V, E)$, a positive integer L, a sensitivity parameter $f \in \mathbb{N}$, with $f = o(\frac{\log n}{\log \log n})$, and an algorithm $\mathcal{A}$ that can build a path-reporting DO$^{\leqslant L}$ with stretch (α, β), space S, query time Q, and preprocessing time T such that (i) both distance and path-reporting queries are deterministic and (ii) when the DO$^{\leqslant L}$ is queried on a pair (s, t), the number of edges of the reported path, if a path is returned, is at most the value of the reported distance.

Output: The data structure of a deterministic f-DSO$^{\leqslant L}$ of G with stretch (α, β).

for $i = 0, \ldots, f$ **do** `// Compute` i`-th level of all FT-trees`
  $\mathcal{HM}'_i \leftarrow \emptyset$;
  foreach $s, t \in V, s \neq t$ **do**
    if $i = 0$ **then** `// Build the root node of` $FT(s,t)$
      Add the root node x to $FT(s,t)$ with $F_x = \emptyset$;
    else `// Build the` i`-th level with` $i > 0$ `from nodes of level` $i-1$
      foreach *node x in $(i-1)$-th level of $FT(s,t)$* **do**
        foreach *edge e in P_x* **do**
          Add to $FT(s,t)$ a child x_e of x with $F_{x_e} = F_x \cup \{e\}$;
    foreach *node x in i-th level of $FT(s,t)$* **do**
      Use Dijkstra or query a precomputed path-reporting f-DSO$^{\leqslant L}$ $\mathcal{O}$ that returns exact distances (*e.g.*, the f-DSO$^{\leqslant L}$ of Theorem 39 in [KP21]), to get a shortest path $P'(s,t,F_x)$ from s to t in $G - F_x$;
      if $|P'(s,t,F_x)| \leqslant L$ **then**
        $P'_x \leftarrow P'(s,t,F_x)$;
        add (P'_x, F_x) to $\mathcal{HM}_i$;
      else
        $P'_x \leftarrow \perp$;
  Compute a family of graphs $\mathcal{G}_i$ that hits-and-misses all pairs in $\mathcal{HM}_i$;
  Use $\mathcal{A}$ to compute the family $\mathcal{O}_i = \{\mathcal{O}_H \mid H \in \mathcal{G}_i\}$;
  Build a data structure $\mathcal{D}_i$ that, when queried on a node x in the i-th level of $FT(s,t)$, with $P'_x \neq \perp$, deterministically returns $\mathcal{O}_x \in \mathcal{O}_i$ that hits-and-misses $(P(s,t,F_x), F_x)$, where $P(s,t,F_x)$ is a path from s to t in $G - F_x$ with at most $\alpha L + \beta$ edges;
  foreach $s, t \in V, s \neq t$ **do**
    foreach *node x in i-th level of $FT(s,t)$, with $P'_x \neq \perp$* **do**
      Query $\mathcal{D}_i$ with x to find the oracle $\mathcal{O}_x \in \mathcal{O}_i$;
      Query $\mathcal{O}_x$ to compute the path $P(s,t,F_x)$;
      $P_x \leftarrow P(s,t,F_x)$;
return $\{(\mathcal{O}_i, \mathcal{D}_i) \mid i \in \{0, ..., f\}\}$;

stretch (α, β) to compute a path P_x from s to t in $G - F_x$, and, finally, we substitute P'_x with P_x in node x. So, at the end of the second stage, each node x contained in the i-th level of an FT-tree stores the path P_x computed using the representative DO$^{\leqslant L}$ $\mathcal{O}_{H_x} \in \mathcal{O}_i$. This means that the FT-trees are built using the paths P_x, and not the paths P'_x that are computed during the first

Algorithm 4: The query algorithm of the f-DSO$^{\leqslant L}$ with stretch (α, β) of Theorem 6.

Input : A triple (s, t, F), where $s, t \in V$, with $s \neq t$ and $F \subseteq E$, $|F| \leqslant f$.
Output: A path from s to t in $G - F$ with at most $\alpha L + \beta$ edges, if $d(s, t, F) \leqslant L$, or $\perp$ otherwise.

$F_x \leftarrow \emptyset$; $i \leftarrow 0$; status $\leftarrow$ not-found;
while *status* = *not-found* **do** `// Emulate the search in` $FT(s,t)$
  Query $\mathcal{D}_i$ with (s, t, F_x) to get the representative DO$^{\leqslant L}$ $\mathcal{O}_x \in \mathcal{O}_i$;
  Query $\mathcal{O}_x$ with (s, t) to compute $P_x = P(s, t, F_x)$;
  if $P_x = \perp$ *or* $|P_x| > \alpha L + \beta$ **then** `// Check for feasibility`
    status $\leftarrow$ not-exist;
  else if $E(P_x) \cap F = \emptyset$ **then**
    status $\leftarrow$ found;
  else
    Let $e \in E(P_x) \cap F$;
    $F_x \leftarrow F_x \cup \{e\}$;
    $i \leftarrow i + 1$;
if status = found **then return** P_x **else return** $\perp$;

stage. Therefore, when we move from level $i < f$ to level $i + 1$, each node x of the i-th level has one child node x_e for each edge e of P_x, and $F_{x_e} = F_x \cup \{e\}$. Therefore, the branching factor of the nodes in the FT-trees is $\alpha L + \beta$ in the worst case.

The query algorithm. We now describe how we can answer to a query (s, t, F), and use the precomputed family of DOs for hop-short distances $\{\mathcal{O}_0, \ldots, \mathcal{O}_f\}$ to emulate the search in $FT(s, t)$ (see Algorithm 4).

To emulate the exploration of a node x in level i of $FT(s, t)$, we use the auxiliary data structure to identify the representative DO$^{\leqslant L}$ in $\mathcal{O}_i$ and then we query it with the pair (s, t) to compute the path P_x. We check whether P_x contains some of the failing edges of F. If P_x does not contain a failing edge, we return it. Otherwise, if P_x contains some of the failing edges F, we select any edge e of F that is also contained in P_x and emulate the traversal of $FT(s, t)$ by emulating the exploration of the child node x_e of x associated with the failure e.

The analysis. In the above description we used multiple times Dijkstra to compute a shortest path P'_x in $G - F_x$ for every node x of the FT-trees, but this adds a term of $\widetilde{O}(n^2(\alpha L + \beta)^f m)$ to the preprocessing. To improve upon the preprocessing time and get rid of the $\widetilde{O}(n^2(\alpha L + \beta)^f m)$ term, we use the path-reporting f-DSO$^{\leqslant L}$ of Theorem 39 in Karthik and Parter [KP21] with parameters L and f during preprocessing, whose preprocessing time is $O(fL \log n)^{f+1} \cdot APSP^{\leqslant L}$, where $APSP^{\leqslant L}$ is the time needed by an algorithm to compute all-pairs shortest paths with at most L edges in a graph H with n vertices and $O(m)$ edges. In our case it is sufficient to use the trivial $\widetilde{O}(mn)$ APSP algorithm as $APSP^{\leqslant L}$, so the time to compute this oracle is $O(fL \log n)^{f+1} \cdot O(mn)$. This oracle answers to a distance query (s, t, F) in time $O(f^2)$, and reports the path $P(s, t, F)$ in additional $O(L)$ time.

We use Theorem 48 of Karthik and Parter to compute the set of graphs $\mathcal{G}_i$ in time $\widetilde{O}(n^2(\alpha L + \beta)^f) \cdot O(fL \log n)^f)$. Lemma 51 of Karthik and Parter states that, for every node x of level i in

our FT-trees, there are $O(f \log n)$ graphs in $\mathcal{G}_i$ that do not contain F_x, and at least one of such $O(f \log n)$ graphs hits-and-misses x. In a similar way as in Section 4.3 of [BCC+23], one can construct a data structure $\mathcal{D}_i$ by specifying a subset of columns of an error-correcting code matrix. The size of the structure is $O(Lf \log n)^f \cdot O(f \log n)$ and given a set F_x of at most f edges, it can compute the (indices of a) set of $O(f \log n)$ graphs that do not contain F_x in time $\widetilde{O}(f^2 L)$.

$\mathcal{A}$ is the construction algorithm of any DO$^{\leqslant L}$ with stretch of (α, β). As mentioned above, we use $\mathcal{A}$ to convert each graph H in $\mathcal{G}_i$ into a corresponding DO$^{\leqslant L}$ $\mathcal{O}_H$ with stretch (α, β).

The auxiliary data structure we use in our theorem to identify the representative DO$^{\leqslant L}$ in $\mathcal{O}_i$ of a given node x of level i in $FT(s,t)$ is defined as follows. We use Lemma 51 in Karthik and Parter to identify the $O(f \log n)$ graphs that do not contain F_x in $\widetilde{O}(f^2 L)$ time. For each of these $O(f \log n)$ graphs, say H, we look at the corresponding DO$^{\leqslant L}$ $\mathcal{O}_H \in \mathcal{O}_i$ with stretch (α, β) and query $\mathcal{O}_H$ with the pair (s,t) to discover the (α, β)-approximate distance $\hat{d}_H(s,t)$ from s to t in the graph H. We point out that, at this particular stage, we only need to retrieve the values of the distances from s to t in the $O(f \log n)$ graphs. Among all distances computed, we take the minimum value and look at the corresponding oracle $\mathcal{O}_H \in \mathcal{O}_i$ associated to a graph $H \in \mathcal{G}_i$. We assume that each oracle in $\mathcal{O}_i$ has a unique identifier. So, in case of ties, we select the DO$^{\leqslant L}$ with minimum identifier. We use exactly the same deterministic algorithm for building the FT-trees, that is, the path P_x that replaces P'_x is the one reported by querying the oracle O_H with the pair (s,t).

We recall that Q, S, and T are, respectively, the query time, space, and preprocessing time of the path-reporting DO$^{\leqslant L}$ of a graph with n vertices and $O(m)$ edges that is built by algorithm $\mathcal{A}$.

Lemma 12. *Algorithm 3 constructs an f-DSO of size $O(fL \log n)^f \cdot \mathsf{S}$ in time $\widetilde{O}(n^2(\alpha L + \beta)^f) \cdot (O(fL \log n)^f + \mathsf{Q}) + O(fL \log n)^f \cdot \mathsf{T}$.*

Proof. First, we preprocess the f-DSO$^{\leqslant L}$ $\mathcal{O}_{KP}$ of Karthik and Parter from Theorem 39 in [KP21] with parameters L and f in $O(fL \log n)^{f+1} \cdot mn$ time, as described above.

Then we build, level by level, n^2 FT-trees $FT(s,t)$. We claim that each FT-tree contains at most $(\alpha L + \beta)^f$ nodes, as every node x of the tree contains either a path $P_x := \perp$ and then x must be a leaf node, or it contains a path P_x with at most $\alpha L + \beta$ edges that is reported by some DO$^{\leqslant L}$ with stretch (α, β) (if we did not find a path with at most $\alpha L + \beta$ edges then we store $P_x := \perp$ as well). The height of an FT-tree is at most f and the number of its nodes is $O((\alpha L + \beta)^f)$.

Next, we analyze the time it takes to construct the i-th level of the fault tolerant trees. Similar to the above claim, we can observe that there are at most $(\alpha L + \beta)^i$ nodes at level i. For each such node x, we first query $\mathcal{O}_{KP}$ with (s,t,F_x) to obtain P'_x in $O(f^2 + L)$ time. Then, we collect all the pairs (P'_x, F_x) for every x in the i-level of any FT-tree $FT(s,t)$, and run the algorithm of Theorem 48 in Karthik and Parter [KP21] which computes a set of $O(fL \log n)^i$ graphs $\mathcal{G}_i$ that hits-and-misses all the pairs (P'_x, F_x) corresponding to the nodes x of that level in $\widetilde{O}(n^2(\alpha L + \beta)^i) \cdot O(fL \log n)^i$ time. Next, we use $\mathcal{A}$ to construct a DO$^{\leqslant L}$ with stretch (α, β) for every graph in $\mathcal{G}_i$ in $O(fL \log n)^i \cdot \mathsf{T}$ time. The data structure $\mathcal{D}_i$ does not require additional construction time as it utilizes data structures we have already constructed above. Finally, for every node x in the i-th level of $FT(s,t)$ with $P'_x \neq \perp$ we query $\mathcal{D}_i$ with x to find the representative oracle $\mathcal{O}_x \in \mathcal{H}$ in $O(f^2 L)$ time, then we query $\mathcal{O}_x$ with (s,t,F_x) in Q time and report the path $P(s,t,F_x)$ in additional $O(\alpha L + \beta)$ time.

In total, the preprocessing time of Algorithm 3 is dominated by $\widetilde{O}(n^2(\alpha L + \beta)^f) \cdot (O(fL \log n)^f + \mathsf{Q}) + O(fL \log n)^f \cdot \mathsf{T}$.

The size of the f-DSO$^{\leqslant L}$ is the size S of the input DO multiplied by the $O(fL \log n)^i$ graphs that are generated in line 11 over all levels $0 \leqslant i \leqslant f$, which sums up to $O(fL \log n)^f \cdot \mathsf{S}$. □

Lemma 13. *Algorithm 4 takes $\widetilde{O}(f \cdot (\mathsf{Q} + (\alpha + f^2)L + \beta))$ time.*

Proof. The algorithm simulates the traversal of the query (s,t,F) in the FT-tree $FT(s,t)$. Starting with $F_x \leftarrow \emptyset$, at each iteration an (α,β)-approximate shortest path P_x from s to t in the graph $G-F'$ is computed by querying $\mathcal{D}_i$ to find the representative $\mathrm{DO}^{\leqslant L}$ $\mathcal{O}_x$ that hits-and-misses (P_x,F_x), and querying $\mathcal{O}_x$ with (s,t) to find the path P_x. The query to $\mathcal{D}_i$ takes $O(f^2L + f\log n \cdot \mathsf{Q})$ time as it includes using the hit-and-miss error correcting codes in $O(f^2L)$ time to find $O(f\log n)$ DOs for hop-short distances that hit-and-miss (P'_x,F_x), querying each of these oracles with (s,t) in $O(f\log n\cdot\mathsf{Q})$ time, and obtaining the representative $\mathrm{DO}^{\leqslant L}$ $\mathcal{O}_x$, which is the first to return the minimum s-to-t distance. Then querying $\mathcal{O}_x$ with the pair (s,t) takes additional Q time and reporting an (α,β)-approximate shortest path P_x using $\mathcal{O}_x$ takes additional $O(\alpha L+\beta)$ time. In total, it takes $O(f^2L + f\log n\mathsf{Q} + \alpha L + \beta)$ to compute the path P_x. If $|P_x| > L$ then the query stops and returns that there is no path of length at most L from s to t in $G-F$. Otherwise, let $e \in E(P_x)\cap F$, the algorithm adds e to F_x and continues to the next iteration.

As $F_x \subseteq F$ and $|F| \leqslant f$, the algorithm must finish after at most $f+1$ iterations, and as each iteration takes $O(f^2L + f\log n\mathsf{Q} + \alpha L + \beta)$ time, in total the runtime of the query is $\widetilde{O}(f\cdot(\mathsf{Q} + (\alpha+f^2)L + \beta))$. □

Lemma 14 (Correctness)**.** *Given a query (s,t,F), Algorithm 4 computes either a path P_x from s to t in $G-F$ with at most $\alpha L+\beta$ edges such that $d(s,t,F) \leqslant |P_x| \leqslant \alpha d(s,t,F)+\beta$ or it returns $\perp$. If $d(s,t,F)\leqslant L$, then the algorithm always returns a path P_x.*

Proof. We prove by induction on $0 \leqslant i \leqslant f$ that Algorithm 4 emulates the search in the FT-tree $FT(s,t)$ that was built by Algorithm 3. This is enough to show correctness.

For $i=0$, i.e., $F_x=\emptyset$, there is only one deterministic data structure $\mathcal{O}\in\mathcal{O}_0$ which is an (α,β)-stretch $\mathrm{DO}^{\leqslant L}$ of G. Querying $\mathcal{O}$ with the pair (s,t) either returns a path P_x from s to t in G such that $|P_x| \geqslant d(s,t)$ or it returns $\perp$. Moreover, if $d(s,t)\leqslant L$, then the oracle indeed returns a path such that $|P_x| \leqslant \alpha d(s,t)+\beta \leqslant \alpha L+\beta$ as it guarantees a stretch of (α,β). As the oracle is deterministic, $|P_x|$ corresponds exactly to the path that was stored in the root of $FT(s,t)$ by Algorithm 3.

For the inductive step, by the induction hypothesis assume that, a node x in level $i-1$ of $FT(s,t)$ stores exactly the same information that was associated with it by Algorithm 3.

The proof breaks into the following two cases.

Case 1: node x stores $\perp$. In this case, it must be that $d(s,t,F_x) > L$. As $F_x\subseteq F$, we have that $d(s,t,F)\geqslant d(s,t,F_x) > L$ and Algorithm 4 correctly returns $\perp$.

Case 2: node x stores a path P_x with at most $\alpha L+\beta$ edges such that $d(s,t,F_x) \leqslant |P_x| \leqslant \alpha d(s,t,F_x)+\beta$. Then, either $E(P_x)\cap F=\emptyset$, and the algorithm correctly returns P_x as $d(s,t,F)\leqslant|P_x|\leqslant\alpha d(s,t,F_x)+\beta\leqslant\alpha d(s,t,F)+\beta$, or $E(P_x)\cap F\neq\emptyset$. In this latter case, let $e\in E(P_x)\cap F$. In line 10 at the end of the $(i-1)$-iteration of Algorithm 4 we add e to F_x. Let x be the node of level i that corresponds to the new F_x. Clearly, Algorithm 3 added this node to $FT(s,t)$. We show that the information computed by Algorithm 4 for x coincides with the information that Algorithm 3 stored in x.

According to Theorem 39 in [KP21], querying their exact f-$\mathrm{DSO}^{\leqslant L}$ with (s,t,F_x) results with a shortest path P'_x from s to t in $G-F_x$. According to Theorem 48 and Lemma 51 in [KP21], the family of graphs $\mathcal{G}_i$ computed in Step 17 of Algorithm 3 has the property that there are $O(f\log n)$ graphs among $\mathcal{G}_i$ which exclude F_x and at least one of these graphs, say G_j, also contains P'_x. Recall that $\mathcal{O}_i$ is the set of the (α,β)-stretch DOs for hop-short distances constructed for all graph is $\mathcal{G}_i$. We query all the DOs for hop-short distances corresponding to the $O(f\log n)$ graphs that exclude F_x ($\mathcal{O}_{\mathcal{G}_j}$ included) with the pair (s,t)

to (α, β)-approximate the value $d(s, t, F - x)$, we keep the minimum-index $\mathrm{DO}^{\leqslant L}$ $\mathcal{O}_x$ that returned the minimum among the computed values as in Algorithm 3, and we query it to report the path $|P_x|$ from s to t in $G - F_x$. As all the queried oracles are deterministic, if all of them ($\mathcal{O}_{\mathcal{G}_j}$ included) return $\perp$, then $d(s, t, F_x) > L$, Algorithm 3 stores $\perp$ in x, and Algorithm 4 correctly returns $\perp$. Therefore, we can assume that $\mathcal{O}_x$ returns a path $|P_x|$. As the graph $\mathcal{G}_x \in \mathcal{G}_i$ corresponding to $\mathcal{O}_x$ excludes F_x, we have that $|P_x|$ is a path from s to t in $G - F_x$. Therefore, $d(s, t, F_x) \leqslant |P_x|$. Let $\hat{d}_x$ and $\hat{d}_j$ be the distances returned by querying $\mathcal{O}_x$ and $\mathcal{O}_j$ with the pair (s, t), respectively. By the choice of $\mathcal{O}_x$ and because $\mathcal{O}_j$ has a stretch of (α, β), we have $\hat{d}_x \leqslant \hat{d}_j \leqslant \alpha d_{\mathcal{G}_j}(s, t, F_x) + \beta = \alpha |P'_x| + \beta$. Furthermore, as the s-to-t path $|P_x|$ reported by $\mathcal{O}_x$ is no longer than $\hat{d}_x$, it follows that $|P_x| \leqslant \alpha |P'_x| + \beta$. As a consequence, if $|P_x| > \alpha L + \beta$, then $|P'_x| > L$ and Algorithm 4 correctly returns $\perp$ as Algorithm 3 stored $\perp$ in x. If $|P_x| \leqslant \alpha L + \beta$, then Algorithm 3 stores P_x in the node x (lines 22 and 23) and, by construction, Algorithm 4 correctly simulates the search in $FT(s, t)$.

□

6 General Distance Sensitivity Oracles

This section proves Theorem 7 (restated below), which takes an f-$\mathrm{DSO}^{\leqslant L}$ for short distances and turns it into a general f-DSO. Let f be the sensitivity of the input DSO and $\widehat{d^{\leqslant L}}(s, t, F)$ the returned estimate with stretch (α, β). Recall that we always assume $\widehat{d^{\leqslant L}}(s, t, F) \geqslant d(s, t, F)$; but $\widehat{d^{\leqslant L}}(s, t, F) \leqslant \alpha \cdot d(s, t, F) + \beta$ only needs to hold for those queries for which s and t have a replacement path on at most L edges. The input data structure can be the one from Theorem 6 but this is not required. In the current setting, we assume that G is undirected, unweighted, and has unique shortest paths. The overall ideas are similar to that of Bilò et al. [BCC+24]. However, their result focus strictly on a multiplicative stretch of 3. When implementing the general ideas, we divert more and more in order to enable our reduction to work with input DSOs with general stretch (α, β). We highlight the key differences at the respective places.

Theorem 7. *Let G be an undirected, unweighted graph with n vertices, m edges, and unique shortest paths. Let f and L be positive integer parameters possibly depending on n and m such that $2 \leqslant f \leqslant L \leqslant n$ as well as $L = \omega(\log n)$. Assume access to an f-edge fault-tolerant distance sensitivity oracle for replacement paths in G with at most L edges, that has stretch (α, β), space S_L, query time Q_L, and preprocessing time T_L. Then, for every $\varepsilon = \varepsilon(n, m, f, L) > 0$, and $\beta = o\left(\frac{\varepsilon^2 L}{f^3 \log n}\right)$, there is a randomized (general) f-edge fault-tolerant distance sensitivity oracle for G that with high probability has*

- *stretch $(\alpha(1+\varepsilon), \beta)$,*
- *query time $\widetilde{O}(f^5 L^{f-1}(\mathsf{Q}_L + f)/\varepsilon^2)$,*
- *preprocessing time $\mathsf{T}_L + O(L^{3f} n) + \widetilde{O}(f^2 m n^2 / L) \cdot O(\log n/\varepsilon)^{f+1}$.*

The space of the data structure is w.h.p.

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

If additionally $f \geqslant 4$ and $L = \widetilde{O}(\sqrt{f^3 m/\varepsilon})$, the data structure can be made deterministic with the same stretch, query time, preprocessing time, and space.

6.1 Preprocessing and Space

The first step of our preprocessing algorithm is to construct and store the assumed input f-DSO$^{\leqslant L}$ with stretch (α, β), taking time T_L and storage space S_L. We then choose an approximation parameter $\varepsilon > 0$ that is used to control the increase of the multiplicative stretch, namely, the resulting DSO has stretch $(\alpha(1+\varepsilon), \beta)$. For technical reasons, we assume $\varepsilon \leqslant 12/7$. Note, however, that we do not require it to be constant.

Define the *granularity* as an integer parameter $\lambda = \lambda(\varepsilon, L)$. Its value will be made precise later,[5] for now it is enough to require $\lambda \leqslant L$. To ease notation, we assume that λ is even; otherwise, every occurrence of $\lambda/2$ below can be replaced by $\lfloor \lambda/2 \rfloor$.

Pivots. After constructing the input DSO for hop-short distances, the preprocessing algorithm samples and stores two random subsets $B_1, B_2 \subseteq V$ of vertices. These sets have vastly differing densities to fulfill their purpose.[6] The input DSO for short distances can estimate a replacement path as long as the endpoints are sufficiently close in $G - F$. For general replacement distances, we use the pivots in B_1 and B_2 to stitch together short paths. We will do so by constructing so-called FT-trees with or without granularity λ between pivots. As it turns out, the FT-tree with granularity have a very low query time (see the discussion after Lemma 20). Unfortunately, this advantage is bought by a much higher space requirement. This is why we choose the set B_1 to be very small and build FT-trees with granularity only between pairs of vertices in $B_1 \times B_1$. However, since there are only few *pivots of the first type*, for a given vertex $u \in V$ there may be none in the vicinity of u if the ball around that vertex in $G - F$ is too sparse. We thus have to provide a fall back option. We sample another set B_2 at a much higher frequency to ensure that there is always some *pivot of the second type* close to u. Since we only need them if the ball around u is sparse, it will be sufficient to store and check *all* pivots from B_2 around u.

Each vertex of V is included in the set B_1 independently with probability $C_1 f \log_2(n)/L^f$ for a constant $C_1 > 0$. An application of standard Chernoff bounds, see e.g. [GV20, RZ12], shows that w.h.p. $|B_1| = \widetilde{O}(fn/L^f)$. Let $u \in V$ and $F \subseteq E$, $|F| \leqslant f$, a set of failing edges and $\mathrm{ball}_{G-F}(u, \lambda/2)$ be the ball of all vertices reachable from u in $G - F$ within $\lambda/2$ hops. We call the ball *dense*, if it has $|\mathrm{ball}_{G-F}(u, \lambda/2)| > L^f$ elements; and *sparse* otherwise. Chernoff bounds also show that, with high probability over all pairs (u, F), whenever some $\mathrm{ball}_{G-F}(u, \lambda/2)$ is dense, it intersects B_1.

The properties of the other set B_2 are a bit more involved. We define the notions of a *trapezoid* and a path being *far away* from all failures as introduced by Chechik et al. [CCFK17].

Definition 15 ($\frac{\varepsilon}{6}$-trapezoid)**.** Let $F \subseteq E$ a set of edges, $u, v \in V$ two vertices, and P a u-v-path in $G - F$. The $\frac{\varepsilon}{6}$*-trapezoid* around P in $G - F$ is

$$\mathrm{tr}^{\varepsilon/6}_{G-F}(P) = \{\, z \in V \backslash \{u, v\} \mid \exists y \in V(P) \colon d_{G-F}(y, z) \leqslant \frac{\varepsilon}{6} \cdot \min(|P[u..y]|, |P[y..v]|) \,\}.$$

The path P is *far away* from all failures in F if $\mathrm{tr}^{\varepsilon/6}_{G-F}(P) \cap V(F) = \emptyset$; otherwise P is *close* to F.

Each vertex is sampled for B_2 independently with probability $C_2 f \log_2(n)/\lambda$ for some constant $C_2 > 0$. We call the elements of B_2 *pivots of the second type*. Again, we have w.h.p. $|B_2| = \widetilde{O}(fn/\lambda)$ such pivots. Moreover, by choosing C_2 sufficiently large, we achieve the following covering property of B_2. With high probability over all triples of vertices $u, v, w \in V$ and failure sets $F \subseteq E$ with $|F| \leqslant f$, the following holds.

[5] The granularity will turn out to be $\lambda = \varepsilon L/9$.

[6] Set B_1 has a low density in V and is used to hit *dense* balls, while B_1 is dense and supposed to hit *sparse* balls instead. To avoid confusion, we do not use this distinction when referring to B_1, B_2.

- If the unique shortest u-v-path in G has at least L edges, then it contains a vertex from B_2.
- If the u-v-replacement path in $G - F$ has at least $\lambda/2$ edges and is far away from all failures in F, then it contains a vertex of B_2.
- If the concatenation of the u-v-replacement path and the v-w-replacement path in $G - F$ has at least λ edges and is far away from all failures in F, then it contains a vertex of B_2 among its first and last $\lambda/2$ vertices.

The second and third item together imply that w.h.p. any concatenation of *at most* two replacement paths that is hop-long and far away from F has a pivot of the second type within distance $\lambda/2$ from either endpoint.

Pivot trees. We need a mechanism in place that allows us to check at query time whether there is a pivot of the first type that is sufficiently close to v; and, if not, gives access to a set of not too many pivots of the second type in the vicinity. The problem is that there are too many graphs $G - F$ to preprocess them directly. Here, we depart from [BCC+24]. They used an (L, f)-replacement path coverings (RPC) [WY13, KP21] as a proxy for the $G-F$. RPCs are families of spanning subgraphs of G such that for each failing set F, $|F| \leqslant f$, and pair of vertices with a replacement path with at most L edges, there exists one graph in the family that contains the path but no edge of F. Their construction realizes this with $(3L)^{f+o(1)}$ graphs.[7] We do not need the full power of replacement paths coverings to find the pivots of the two types. We give a much simpler solution based on what we call *pivot trees*. The advantage is that they only need to be computed for the original graph G. Even when incorporating the size of the pivot trees, this allows us to reduce the space for this part of the data structure by roughly a factor $3^{f+o(1)}L^{1+o(1)}$.

Consider $\mathrm{ball}_G(u, \lambda/2)$ around u in the original graph G. It can be computed using a breath-first search. We modify it to stop as soon at it finds the first pivot from B_1 (if there is one). Note that $\mathrm{ball}_G(u, \lambda/2) \cap B_1 = \emptyset$ implies $|\mathrm{ball}_G(u, \lambda/2)| \leqslant L^f$ w.h.p. That means, the BFS explores all of $\mathrm{ball}_G(u, \lambda/2)$ in time $O(L^{2f})$ or finds a pivot of the first type even before that.

Suppose first that the search does not find such a pivot. With high probability, the set $\mathrm{ball}_G(u, \lambda/2)$ is sparse and contains at most $\widetilde{O}(fL^f/\lambda)$ pivots of the *second* type from B_2. We store this set $\mathrm{ball}_G(u, \lambda/2) \cap B_2$. Now assume that we have $\mathrm{ball}_G(u, \lambda/2) \cap B_1 \neq \emptyset$. Let $p_G(u) \in B_1$ be the pivot closest to u that was found by the search. Let P the shortest u-$p_G(u)$-path in G. It has at most $\lambda/2$ edges. For each $e \in E(P)$, we create a child node in which we conduct the same computation for the ball $\mathrm{ball}_{G-e}(u, \lambda/2)$. Note that the new closest pivot $p_{G-e}(u) \in B_1$ may not be the same as $p_G(u)$. This is the major difference to the trees used in Section 5 that always store paths between the same two endpoints. It still holds that the shortest u-$p_{G-e}(u)$-path in $G-e$ has at most $\lambda/2$ edges. We iterate this construction until the pivot tree reached depth f. In each child node, a new edge from the previous path from u to its closest pivot is removed.

We build such a pivot tree for every $u \in V$. Each one has $O(\lambda^f)$ nodes. They store either a path with $\lambda/2$ edges or some set $\mathrm{ball}_{G-F}(u, \lambda/2) \cap B_2$. Using $2 \leqslant f$ and $\lambda \leqslant L$, we get $\lambda/2 = \widetilde{O}(fL^f/\lambda)$. The total size of all trees is thus $\widetilde{O}(f\lambda^{f-1}L^f n)$. They can be computed in time $O(\lambda^f L^{2f} n)$.

Additionally, for every pivot $p \in B_2$ of the second type, the preprocessing computes the shortest-path tree T_p rooted at p in the original graph G. For each T_p, the data structure of Bender and Farach-Colton [BF00] is constructed that supports lowest common ancestor (LCA) queries in constant time. Preparing all LCA-data structures together takes time $O(|B_2|\,(m+n)) = \widetilde{O}(fmn/\lambda)$ and storing them takes space $O(|B_2|\,n) = \widetilde{O}(fn^2/\lambda)$.

[7] This needs some reconstruction of the results in [BCC+24, Section 4.2 & 5.3] as they use constant f.

W.h.p. the preprocessing algorithm spent time $\mathsf{T}_L + O(\lambda^f L^{2f} n) + \widetilde{O}(fmn/\lambda)$ so far and stored information taking up space

$$\mathsf{S}_L + |B_1| + |B_2| + \widetilde{O}(f\lambda^{f-1}L^f n) + \widetilde{O}\left(f\frac{n^2}{\lambda}\right) = \mathsf{S}_L + \widetilde{O}(f\lambda^{f-1}L^f n) + \widetilde{O}\left(f\frac{n^2}{\lambda}\right).$$

Fault-tolerant trees again. We have already utilized the fault-tolerant trees of Chechik et al. [CCFK17] in our proof of Theorem 6. There, every edge of the path included in a node formed its own segment. The main construction in [CCFK17] extended this to segments of exponentially increasing length. Bilò et al. [BCC+24] later introduced a hybrid form in which the first and last λ edges are on their own and longer segments only appear in the middle part. We build on the latter version. The main idea behind the switch from single edges to longer segments is to reduce the storage space of an FT-tree. Any segment is going to be identified by its endpoints, so reducing their number lowers the space. However, this also leads to a more coarse-grained picture at query time as failing whole segments can lead to much more than f failures overloading the underlying f-DSO. The segments thus need to be structured in a way that even if they fail completely certain paths are still guaranteed to survive. We make this statement precise in Lemma 21 after describing the query algorithm in Section 6.2.

For the construction, we first have to define certain types of concatenated paths. Afek, Bremler-Barr, Kaplan, Cohen, and Merritt [ABK+02, Theorems 1 & 2] showed that every replacement path in $G - F$ consists of at most $|F| + 1$ shortest paths of the original graph G, possibly interleaved with $|F|$ edges in case G is weighted. We call a path of this structure $|F|$-*decomposable*.

Definition 16 (ℓ-decomposable paths)**.** If G is unweighted, an *ℓ-decomposable path* is a concatenation of at most $\ell + 1$ shortest paths of G. If G is edge-weighted, an *ℓ-decomposable path* may have an additional edge between any pair of consecutive constituting shortest paths.

It is easy to see that any ℓ-decomposable path is also ℓ'-decomposable for any $\ell' \geqslant \ell$. Also, any subpath of an ℓ-decomposable path (with granularity λ) is again ℓ-decomposable The $(2f+1)$-decomposable paths are of special interest to us since they comprise all concatenations of up to two replacement paths in $G - F$. Let $A \subseteq E$ be a set of edges, possibly much more than f, and let $s, t \in V$ be two vertices. We use $d^{(2f+1)}(s, t, A)$ to denote the length of the shortest $(2f+1)$-decomposable path from s to t in $G - A$; or $+\infty$ if no such path exists.

Definition 17 (ℓ-expath with granularity λ)**.** An *ℓ-expath with granularity λ* 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 the i-th ℓ-decomposable path has at most $\min(2^i, 2^{2\log_2(n)-i})$ edges.

Definition 17 includes the case where some or all ℓ-decomposable paths in P_b are empty, only the maximum number of edges is bounded. Bilò et al. [BCC+24, Section 7] gave and $\widetilde{O}(fm)$-time algorithm to compute the shortest $(2f+1)$-expath with granularity λ in any subgraph of G when given access to the original pair-wise distances in G. They come annotated with the constituting $(2f+1)$-decomposable paths and, in turn, with their shortest paths in G (and interleaving edges).

The last things we need before defining FT-trees are that of netpoints, segments, and parts.

Definition 18 (Netpoints with granularity λ)**.** Let $P = (v_1, v_2, \ldots, v_\ell)$ be a path. If $|P| \leqslant \lambda$, then the *netpoints of P with granularity λ* are all vertices in $V(P)$. Otherwise, define p_{left} to be all pairs of consecutive vertices $v_j, v_{j+1} \in V(P)$ with $\frac{\lambda}{2} \leqslant j \leqslant \ell - \frac{\lambda}{2}$ for which there exists an

integer $i \geqslant 0$ such that $|P[v_{\frac{\lambda}{2}}..v_j]| < (1+\frac{\varepsilon}{24})^i \leqslant |P[v_{\frac{\lambda}{2}}..v_{j+1}]|$. Analogously, let p_{right} be all vertices $v_j, v_{j-1} \in V(P)$ such that $|P[v_j..v_{\ell-\frac{\lambda}{2}}]| < (1+\frac{\varepsilon}{24})^i \leqslant |P[v_{j-1}..v_{\ell-\frac{\lambda}{2}}]|$ for some i. The *netpoints of P with granularity λ* are all vertices in $\{v_1, \dots, v_{\frac{\lambda}{2}}\} \cup p_{\text{left}} \cup p_{\text{right}} \cup \{v_{\ell-\frac{\lambda}{2}}, \dots, v_\ell\}$.

The intuition is as follows. The first and last $\lambda/2$ vertices on the path P are always netpoints. Now consider the central subpath $P[v_{\frac{\lambda}{2}}..v_{\ell-\frac{\lambda}{2}}]$. Let x be a power of $1+\frac{\varepsilon}{24}$. Among all vertices on the subpath that have distance at most x from $v_{\frac{\lambda}{2}}$ mark the one furthest away as well as its successor. Doing this for all possible powers x gives the vertices in p_{left}. The same marking scheme starting from $v_{\ell-\frac{\lambda}{2}}$ on the other end of the subpath gives p_{right}.

Definition 19 (Segments and parts)**.** Let P be any path. A *segment* of P is a subpath between consecutive netpoints with granularity λ. For any edge $e \in E(P)$, the segment of P containing e is denoted $\text{seg}_\lambda(e, P)$. Suppose P is an $(2f+1)$-expath with granularity λ. Its defining subpath P_b consists of $2\log_2(n) + 1$ $(2f+1)$-decomposable paths that in turn each consists of up to $2f+1$ shortest paths P_i. A *part* of P is a maximal subpath of any of the P_i that does not cross netpoints.

The first and last $\lambda/2$ edges of a path each form their own segment. In the central subpath, there are single-edge segments, but also longer ones whose length grows exponential in $1+\frac{\varepsilon}{24}$. However, marking the netpoints both ends of the subpath ensures that these do not become too large. The subdivision of the segments into parts is to align these building blocks with the structure of expaths. This is of course only necessary for segments with more than one edge.

We use $[x, y]$ for a part that reaches from vertex x to y. By definition, $[x, y]$ is the unique shortest x-y-path in G. There are fewer then $\lambda + 2\log_{1+\frac{\varepsilon}{24}}(n) = \lambda + O(\log(n)/\varepsilon)$ segments on any path. Moreover, there exists a universal constant $D > 0$ such that any $(2f+1)$-expath with granularity λ has at most $\lambda + Df\log_2(n)/\varepsilon$ many parts. We sometime also use an alternative upper bound on the number of parts that is independent of $\varepsilon > 0$. Namely, it is $\widetilde{O}(fn)$ since each of the $O(\log n)$ decomposable paths in P_b consists of $O(f)$ shortest (i.e., simple) paths.

We finally turn to the fault-tolerant trees with granularity λ. Each such tree belongs to a pair of vertices $u, v \in V$. In turn, each node ν in the tree $FT_\lambda(u, v)$ is associated with a path P_ν, which is a shortest $(2f+1)$-expath with granularity λ from u to v in the graph $G - A_\nu$, where $A_\nu \subseteq E$ is a set of edges (with cardinality possibly much more than f). We have $A_\nu = \emptyset$ in the root. The path P_ν is computed with the algorithm of Bilò et al. [BCC+24] and, with additional linear scan, annotated with its netpoints, segments, and parts. For each part $[x, y]$, we store two pointers. One to the closest netpoint that comes before x (including x itself), and one to the closest netpoint after y (including). The further processing of $[x, y]$ depends on the number of its edges. If it has more than L edges it contains some pivot $p \in B_1$ w.h.p., where we use that $L \geqslant \lambda$. We store (the ID of) p with $[x, y]$. Otherwise, if $[x, y]$ has at most L edges, we store the original graph distance $d(x, y)$. Computing and post-processing P_ν takes time $\widetilde{O}(fm) + O(n) + \widetilde{O}(fn) = \widetilde{O}(fm)$. Here, the third term is the alternative upper bound on the number of parts. The stored information is $O(\lambda + f\log(n)/\varepsilon)$, a constant number of words per part.

To construct the FT-tree with granularity, we create a child node μ of ν for each segment S of P_ν. We set the new set of edges in the child as $A_\mu = A_\nu \cup E(S)$. We continue with this process until depth f. If the vertices u and v become disconnected in one of the graphs $G - A_\nu$, we mark this by ν being a leaf that does not store any path. In total, the tree $FT_\lambda(u, v)$ has $O(\lambda^f) + O(\log n/\varepsilon)^f$ nodes and thus be computed in time $\widetilde{O}(fm) \cdot (O(\lambda^f) + O(\frac{\log n}{\varepsilon})^f)$ and stored using space $(\lambda + O(f\frac{\log n}{\varepsilon})) \cdot (O(\lambda^f) + O(\frac{\log n}{\varepsilon})^f)$. We leave the expressions as they are for now. They will be simplified after fixing λ and the number of trees.

The description above holds verbatim also for granularity 0 instead of λ,. We refer to this as FT-trees *without granularity* etc.

Finishing the preprocessing. From now on, we limit the granularity to $\lambda = \Theta(\varepsilon L)$. Recall that w.h.p. we have $\widetilde{O}(fn/L^f)$ pivots of the first type. We construct an FT-tree with granularity λ for every pair in $B_1 \times B_1$. Further, an FT-tree without granularity is built for each pair a pivot of the second type and an arbitrary vertex, that is, for $B_2 \times V$. W.h.p. the former trees take space

$$
\begin{aligned}
&|B_1|^2 \cdot \left(\lambda + O\left(f\frac{\log n}{\varepsilon}\right)\right)\left(O(\lambda^f) + O\left(\frac{\log n}{\varepsilon}\right)^f\right)\\
&= \widetilde{O}\left(f^2\frac{n^2}{L^{2f}}\right)\cdot\left(O(\lambda^{f+1}) + \lambda\cdot O\left(\frac{\log n}{\varepsilon}\right)^f + O\left(f\lambda^f\right)\cdot O\left(\frac{\log n}{\varepsilon}\right) + \widetilde{O}(f)\cdot O\left(\frac{\log n}{\varepsilon}\right)^{f+1}\right)\\
&= \widetilde{O}\left(f^2\frac{n^2\lambda^{f+1}}{L^{2f}}\right) + \widetilde{O}\left(f^2\frac{n^2\lambda}{L^{2f}}\right)O\left(\frac{\log n}{\varepsilon}\right)^f + \widetilde{O}\left(f^3\frac{n^2\lambda^f}{L^{2f}}\right)O\left(\frac{\log n}{\varepsilon}\right) + \widetilde{O}\left(f^3\frac{n^2}{L^{2f}}\right)O\left(\frac{\log n}{\varepsilon}\right)^{f+1}\\
&= \widetilde{O}\left(f^2\frac{n^2}{L^{f-1}}\right) + \widetilde{O}\left(f^2\frac{n^2}{L^{2f-1}}\right)O\left(\frac{\log n}{\varepsilon}\right)^f + \widetilde{O}\left(f^2\frac{n^2}{L^{f-1}}\right)O\left(\frac{\log n}{\varepsilon}\right) + \widetilde{O}\left(f^2\frac{n^2}{L^{2f-1}}\right)O\left(\frac{\log n}{\varepsilon}\right)^{f+1}.
\end{aligned}
$$

We used $\lambda \leqslant L$, $\lambda = \Omega(\varepsilon L)$, and $f \leqslant L$ for the transformations in the last line. For the FT-trees between pair in $B_2 \times V$ the space is

$$
n|B_2|\cdot O\left(f\frac{\log n}{\varepsilon}\right)O\left(\frac{\log n}{\varepsilon}\right)^f = \widetilde{O}\left(f^2\frac{n^2}{\lambda}\right)\cdot O\left(\frac{\log n}{\varepsilon}\right)^{f+1} = \widetilde{O}\left(f^2\frac{n^2}{L}\right)\cdot O\left(\frac{\log n}{\varepsilon}\right)^{f+2}.
$$

We assumed $f \geqslant 2$, which gives $n^2/L^{f-1} \leqslant n^2/L$. Therefore, all terms for both sets of FT-trees are dominated by $\widetilde{O}(f^2n^2/L)\cdot O(\log n/\varepsilon)^{f+2}$.

Recall that we also store the DSO for hop-short distances, the pivot trees, and the LCA data structures. The latter is also dominated by the FT-trees. The total space is

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

The statement in Theorem 7 follows from this via $\lambda \leqslant L$.

A similar calculation shows that the construction time for the FT-trees is

$$
\begin{aligned}
&\widetilde{O}(fm)\cdot\left(|B_1|^2\cdot\left(O(\lambda^f) + O\left(\frac{\log n}{\varepsilon}\right)^f\right) + n|B_2|\cdot O\left(\frac{\log n}{\varepsilon}\right)^f\right)\\
&\quad= \widetilde{O}(fm)\cdot\left(\widetilde{O}\left(f^2\frac{n^2}{L^f}\right) + \widetilde{O}\left(f^2\frac{n^2}{L^{2f}}\right)O\left(\frac{\log n}{\varepsilon}\right)^f + \widetilde{O}\left(f\frac{n^2}{\lambda}\right)O\left(\frac{\log n}{\varepsilon}\right)^f\right)\\
&\quad= \widetilde{O}(fm)\cdot\widetilde{O}\left(f\frac{n^2}{\lambda}\right)O\left(\frac{\log n}{\varepsilon}\right)^f = \widetilde{O}\left(f^2\frac{mn^2}{L}\right)O\left(\frac{\log n}{\varepsilon}\right)^{f+1}.
\end{aligned}
$$

The last summary uses $\lambda = \Omega(\varepsilon L)$ again. Considering the preprocessing of the input DSO for hop-short distances, the pivot trees, and LCA structures, the resulting time is $\mathsf{T}_L + O(\lambda^f L^{2f} n) + \widetilde{O}(f^2mn^2/L)\cdot O(\log n/\varepsilon)^{f+1} = \mathsf{T}_L + O(L^{3f}n) + \widetilde{O}(f^2mn^2/L)\cdot O(\log n/\varepsilon)^{f+1}$.

6.2 Query Algorithm and Time

The resulting f-DSO (for arbitrary distances) is queried with a triple (s,t,F) with $s,t \in V$ being two vertices and $F \subseteq E$ is a set of at most f edges. The task is to approximate the replacement

distance $d(s,t,F)$, which is the distance between s and t in $G-F$. We first describe the high-level algorithm, before going into the details. For the further description, let $u,v \in V$ be any two vertices. Recall that we use $\widehat{d^{\leqslant L}}(u,v,F)$ for the estimate of the u-v-distance in $G-F$ when restricted to paths with at most L edges. It is reported by the short-distance DSO in time Q_L. In this section, we further use $FT_\lambda(u,v,F)$ for the result we get from querying the FT-tree with granularity $FT_\lambda(u,v)$ with the set F. Analogously, we use $FT_0(u,v,F)$ for the value of an FT-tree without granularity. We will see later that the query time for FT-trees with or without granularity, which we respectively denote by $\mathsf{Q}_{FT}^{(\lambda)}$ and $\mathsf{Q}_{FT}^{(0)}$, are asymptotically larger than Q_L and than f^2. We use this fact for simplifications in the O-notation below.

When given a query (s,t,F), an auxiliary weighted complete graph H on the vertex set $V(H) = \{s,t\} \cup V(F)$ is constructed, where $V(F)$ is the set of endpoints of edges in F. For each $\{u,v\} \in E(H) = \binom{V(H)}{2}$, the query algorithm computes a weight $w_H(u,v)$. The eventual answer of our DSO is the distance $d_H(s,t)$ in H.

Computing the weights $w_H(u,v)$. The query algorithm computes $w_H(u,v)$ with one of two methods, depending on whether there are pivots of the first type available in the vicinity of u and v in $G-F$. The respective pivot trees are used for this decision. It is checked, whether the stored path P from u to the closest pivot $p_G(u) \in B_1$ contains any edge of F. If not, then $p_G(u) \in \mathrm{ball}_{G-F}(u,\lambda/2)$ and it is also the closest pivot of the first type in $G-F$. Otherwise, we recurse to an arbitrary child node corresponding to some edge in $E(P) \cap F$. After at most f levels, the algorithm either learns a pivot from B_1 that is close to u in $G-F$, or it arrives in a leave node corresponding to some $F' \subseteq F$ such that $\mathrm{ball}_{G-F'}(u,\lambda/2)$ is too sparse to intersect B_2. In the former case, let $p_u \in B_1$ be the closest pivot. In the latter case, the query algorithm gets access to the pivots of the second type in $\mathrm{ball}_{G-F'}(u,\lambda/2) \cap B_2$. Note that we have $\mathrm{ball}_{G-F'}(u,\lambda/2) \cap B_2 \supseteq \mathrm{ball}_{G-F}(u,\lambda/2) \cap B_2$. That means, all pivots of the second type that are close to u are recovered (and maybe some more). Each check for the emptyness of $E(P) \cap F$ takes time $O(f)$, so the whole time spend in the pivot tree is $O(f^2)$. The query algorithm does the same for the other vertex v. We say the computation of the weight $w_H(u,v)$ is in the *dense ball case* if two pivots $p_u, p_v \in B_1$ are obtained. Otherwise, we say it is in the *sparse ball case*.

First, assume we are in the dense ball case. We stored an FT-tree with granularity λ for the pair $(p_u,p_v) \in B_2^2$. Recall that we use $FT_\lambda(p_u,p_v,F)$ for the value returned by that FT-tree when queried with the failure set F. The weight is defined as

$$w_H(u,v) = \min\Big(\widehat{d^{\leqslant L}}(u,v,F),\ FT_\lambda(p_u,p_v,F) + \lambda\Big)\,. \tag{1}$$

It is computable in time $O(f^2 + \mathsf{Q}_L + \mathsf{Q}_{FT}^{(\lambda)}) = O(\mathsf{Q}_{FT}^{(\lambda)})$.

In the sparse ball case, there exists a vertex $x \in \{u,v\}$ such that the algorithm recovered $\mathrm{ball}_{G-F'}(u,\lambda/2) \cap B_2$. Let $y \in \{u,v\}\backslash\{x\}$ be the other endpoint. The weight is

$$w_H(u,v) = \min\left(\widehat{d^{\leqslant L}}(u,v,F), \min_{p \in \mathrm{ball}_{G-F'}(u,\lambda/2)\cap B_2} \left(\widehat{d^{\leqslant L}}(x,p,F) + FT_0(p,y,F)\right)\right). \tag{2}$$

That means, the query algorithm minimizes the sum $\widehat{d^{\leqslant L}}(x,p,F) + FT_0(p,y,F)$ over all pivots p of the second type in the sparse ball around x, it then compares the result with the estimate $\widehat{d^{\leqslant L}}(u,v,F)$ of the input DSO for hop-short distances. The edge weight is the smaller of the two values. Since w.h.p. $|\,\mathrm{ball}_{G-F'}(u,\lambda/2) \cap B_2| = O(fL^f/\lambda)$, the weight can be obtained in time $\widetilde{O}(f^2 + \mathsf{Q}_L + f\frac{L^f}{\lambda}(\mathsf{Q}_L + \mathsf{Q}_{FT}^{(0)})) = O(f\frac{L^f}{\lambda}\mathsf{Q}_{FT}^{(0)})$.

The auxiliary graph H has $O(f^2)$ edges, so obtaining all weights takes $\widetilde{O}(f^2(\mathsf{Q}_{FT}^{(\lambda)} + f\frac{Lf}{\lambda}\mathsf{Q}_{FT}^{(0)}))$ time. This is also the order of the whole query time of our general DSO since the $\widetilde{O}(f^2)$ time to compute the distance $d_H(s,t)$ with Dijkstra's algorithm is immaterial.

Querying the FT-trees. So far, we used the query procedure of the FT-trees as a black box. We now describe how the value $FT_\lambda(u,v,F)$ is computed, the case without granularity follows immediately by setting $\lambda = 0$. The overall structure of the query algorithm is similar to [BCC+24], but we generalize it to incorporate the stretch (α,β) of the the underlying DSO for hop-short distances. Recall that D is the universal constant such that any $(2f+1)$-expath with granularity λ has at most $\lambda + Df\log_2(n)/\varepsilon$ many parts.

Lemma 20. *Let ν be a node of the FT-tree $FT_\lambda(u,v)$ in which an $(2f+1)$-expath P_ν from u to v is stored. There is an algorithm that either certifies that $d(u,v,F) \leqslant \alpha\cdot|P_\nu| + Df\frac{\log_2(n)}{\varepsilon}\cdot\beta$ or finds the segment $\mathrm{seg}_\lambda(e,P_\nu)$ for some edge $e \in F\cap E(P_\nu)$. The algorithm runs in time $O(f\lambda + f\frac{\log n}{\varepsilon}(\mathsf{Q}_L + f))$.*

Proof. The algorithm first probes each parts of P_ν, whether it can certify that it contains an edge from F. Recall that any part $[x,y]$ is the unique shortest path from x to y in the original graph G.

If the part consists of only a single edge, it is trivial to check whether $\{x,y\} \in F$ in time $O(f)$. In particular, this is the case for the first and last $\lambda/2$ edges of P_ν. If $[x,y]$ has at most L edges, we stored the original distance $d(x,y)$ with it. The algorithms queries the underlying DSO for hop-short distances to get the estimate $\widehat{d^{\leqslant L}}(x,y,F)$. The oracle has stretch (α,β). If $\widehat{d^{\leqslant L}}(x,y,F) > \alpha\cdot d(x,y) + \beta$, then the part must contain a failing edge.

Otherwise, $[x,y]$ has more than L edges. We stored a pivot $p \in B_2$ of the second type that lies on $[x,y]$. By the uniqueness of paths $[x,y]$ is the concatenation of the unique shortest path from x to p in G and the one from p to y. Using the LCA data structure for the shortest-path tree T_p rooted at p, the algorithm can check in time $O(f)$ whether any edge $e \in F$ also lies on $[x,y]$.

Let $\mathcal{P}$ be the collection of all parts of P_ν that are only a single edge, $\mathcal{L}$ the ones with more than L edges, and $\mathcal{R}$ the remaining parts. The path P_ν is the concatenation of the parts in $\mathcal{P}\cup\mathcal{L}\cup\mathcal{R}$ (in a suitable order). Let instead P be the path in which the parts in $\mathcal{P}\cup\mathcal{L}$ remain the same, but each part $[x,y]\in\mathcal{R}$ is exchanged by the replacement path $P(x,y,F)$ in $G-F$.

Checking all the at most $\lambda + O(\log(n)/\varepsilon)$ many parts takes total time $O(f\lambda + f(\log n)(\mathsf{Q}_L+f)/\varepsilon)$. If a part $[x,y]$ is found to contain an edge from F, the corresponding segment is given by the pointer to the closest netpoints before x and after y. Note that we do not need to know the exact failing edge since the whole part lies in the same segment. Otherwise, the algorithm verified that the path P defined above lies in $G-F$ and has length

$$|P| = \sum_{[x,y]\in\mathcal{P}\cup\mathcal{L}} d(x,y) + \sum_{[x,y]\in\mathcal{R}} d(x,y,F) \leqslant \sum_{[x,y]\in\mathcal{P}\cup\mathcal{L}} d(x,y) + \sum_{[x,y]\in\mathcal{R}} (\alpha\cdot d(x,y)+\beta)$$
$$\leqslant \left(\sum_{[x,y]\in\mathcal{P}\cup\mathcal{L}\cup\mathcal{R}} \alpha\cdot d(x,y)\right) + |\mathcal{R}|\beta \leqslant \alpha|P_\nu| + Df\frac{\log_2(n)}{\varepsilon}\cdot\beta. \quad \square$$

After this setup, we describe the query algorithm of FT-trees. When queried with the failure set F, $FT_\lambda(u,v)$ is traversed starting with the root. In each node ν, the algorithm first checks whether u and v are connected, that is, whether a path P_ν is present. If there is no path, the query returns with the answer $+\infty$. Otherwise, the algorithm from Lemma 20 is run with set F. If a segment with a failing edge is found, the traversal recurses on the child node of ν that corresponds to that segment; otherwise, the value $\alpha|P_\nu| + Df\frac{\log_2(n)}{\varepsilon}\beta$ is reported. If a leaf ν^* of $FT_\lambda(u,v)$ is reached that way in which a path $P_{\nu*}$ is stored, the output is $|P_{\nu^*}|$.

The height of the tree $FT_\lambda(u,v)$ is f, so the total query time is $\mathsf{Q}_{FT}^{(\lambda)} = O(f^2\lambda + f^2\frac{\log n}{\varepsilon}(\mathsf{Q}_L+f))$, or $\mathsf{Q}_{FT}^{(0)} = O(f^2\frac{\log n}{\varepsilon}(\mathsf{Q}_L+f))$ without granularity. From $\lambda = \Theta(\varepsilon L)$ and $f \geqslant 2$, we get that $f\frac{L^f}{\lambda}\mathsf{Q}_{FT}^{(0)}$ dominates $\mathsf{Q}_{FT}^{(\lambda)}$, Therefore, the total query time of our (general) DSO is

$$\widetilde{O}\left(f^2\left(\mathsf{Q}_{FT}^{(\lambda)} + f\frac{L^f}{\lambda}\mathsf{Q}_{FT}^{(0)}\right)\right) = \widetilde{O}\left(f^3\frac{L^f}{\lambda}\mathsf{Q}_{FT}^{(0)}\right)$$
$$= \widetilde{O}\left(f^5\frac{L^f}{\lambda}\frac{\log n}{\varepsilon}(\mathsf{Q}_L+f)\right) = \widetilde{O}\left(f^5\frac{L^{f-1}}{\varepsilon^2}(\mathsf{Q}_L+f)\right).$$

Intuitively, a parent-child traversal during the query of an FT-tree simulates the failure of the whole segment $\mathrm{seg}_\lambda(e,P_\lambda)$ instead of only the single edge $e \in F$. Storing information only for the segments, or more accurately for the parts, reduces the space needed to store an FT-tree but in turn makes the query answer less precise. Let ν be the node that produces the output. The set A_ν of edges that are absent in ν may be much larger than F. However, due to definition of segments, the elements in A_ν are heavily clustered. Therefore, if a path is *decomposable and far away from* F (see Definitions 15 and 16) it even avoids all of A_ν. More precisely, the FT-trees with their associated query algorithm have the following key property [BCC+24, Lemmas 5.9 & 6.6]

Lemma 21 ([BCC+24])**.** *Let $u,v \in V$ be two vertices and $F \subseteq E$ a set of at most f edges, and P be the shortest $(2f+1)$-decomposable u-v-path that is far away from F.*

(i) Let $p_u \in \mathrm{ball}_{G-F}(u,\lambda/2) \cap B_1$ and $p_v \in \mathrm{ball}_{G-F}(v,\lambda/2) \cap B_1$, and ν be the output-node of $FT_\lambda(p_u,p_v)$ when queried with F. Then, the path $P(p_u,u,F) \circ P \circ P(v,p_v,F)$ exists in $G-A_\nu$.

(ii) Let ν' be the output-node of $FT_0(u,v)$ when queried with F. Then, the path P exists in $G-A_{\nu'}$.

Observe that Lemma 21 *(i)* applies to the tree $FT_\lambda(p_u,p_v)$ although its construction and query algorithm is completely independent of the endpoints u and v of the path P. The only relation is that p_u and u (respectively, p_v and v) are connected by a path on at most $\lambda/2$ edges in $G-F$. The granularity λ of $FT_\lambda(p_u,p_v)$ means that the first and last $\lambda/2$ edges of any path P_ν stored in a node ν of the FT-tree form their own segment. Failing a segment in this pre-/suffix is equivalent to failing a single edge. Intuitively, this cannot affect paths that lie within $\mathrm{ball}_{G-F}(u,\lambda/2)$ (respectively, in $\mathrm{ball}_{G-F}(v,\lambda/2)$). Conversely, for the exponentially increasing segments in the middle of P_ν, the safety area that ensures the survival of the path P is given by the trapezoid $\mathrm{tr}_{G-F}^{\varepsilon/6}(P)$. If this area is free of failures, meaning that P is far away from F, no segment $\mathrm{seg}_\lambda(e,P_\lambda)$ can reach P. Making this intuition rigorous, however, requires some technical machinery. This is the reason to use expaths P_ν (instead of mere decomposable paths) and for the constants in the definition of trapezoids (Definition 15) and netpoints (Definition 18) differing by a multiplicative factor 4. The details are discussed in [BCC+24].

6.3 Stretch

We are left to show that our DSO has stretch $(\alpha(1+\varepsilon),\beta)$, where (α,β) is the stretch of the input DSO for hop-short distances and $\varepsilon > 0$ is the approximation parameter chosen at the beginning of the preprocessing. The fine-tuned analysis in this section is the main improvement over [BCC+24]. It allows us to generalize f-DSOs to the setting of a positive additive stretch $\beta > 0$ and multiplicative stretch $\alpha \neq 3$.

We first prove bounds on the values $\widehat{d^{\leqslant L}}(x,y,F)$, $FT_0(x,y,F)$, and $FT_\lambda(x,y,F)$ that are used in the computation of the weight $w_H(u,v)$. We then combine all the intermediate bounds in Lemma 27 to prove the stretch. We start by showing that none of the different values underestimate the true replacement distance.

Lemma 22. *Let $u,v,x,y \in V$ be vertices. Then, it holds that*

(i) $\widehat{d^{\leqslant L}}(x,y,F), FT_0(x,y,F), FT_\lambda(x,y,F) \geqslant d(x,y,F)$;

(ii) $w_H(u,v) \geqslant d(u,v,F)$.

Proof. We first prove *(i)*. This is immediate for $\widehat{d^{\leqslant L}}(x,y,F)$ as the estimate for the hop-bounded replacement distance is always at least the true replacement $d(x,y,F)$.

If the FT-tree $FT_\lambda(x,y)$, when queried with set F, answers $+\infty$, this is clearly at least the replacement distance. Now suppose the query procedure stops in some node ν and the value $\alpha \cdot |P_\nu| + Df\frac{\log_2(n)}{\varepsilon} \cdot \beta$ is returned. Then, the algorithm in Lemma 20 certifies that this value is at least $d(x,y,F)$. Otherwise, the query reaches a leaf ν^* at depth f. The tree traversal thus found a segment for each failing edge in F. The path P_{ν^*} from x to y is thus disjoint from F, giving $FT_\lambda(x,y,F) = |P_{\nu^*}| \geqslant d(x,y,F)$. The same argument also holds for $FT_0(x,y,F)$.

We now turn to *(ii)*. First, assume that $w_H(u,v)$ is computed by Equation (1) using the pivots $p_u \in \mathrm{ball}_{G-F}(u,\lambda/2) \cap B_1$ and $p_v \in \mathrm{ball}_{G-F}(v,\lambda/2) \cap B_1$. We need to show that $FT_\lambda(p_u,p_v,F)+\lambda \geqslant d(u,v,F)$. If p_u and p_v are disconnected in $G-F$, we have $FT_\lambda(p_u,p_v,F) = +\infty$ by *(i)* and we are done. Otherwise, there exists some replacement path $P(p_u,p_v,F)$. Let P_{p_u} be the shortest path from u to p_u in $G-F$, and P_{p_v} the shortest path from p_v to v in $G-F$. Note that both paths have at most $\lambda/2$ edges by the choice of p_u,p_v. In summary, $P_{p_u} \circ P(p_u,p_v,F) \circ P_{p_v}$ is some path from u to v in $G-F$. From Part *(i)*, we get that

$$FT_\lambda(p_u,p_v,F) + \lambda \geqslant \lambda/2 + d(p_u,p_v,F) + \lambda/2 \geqslant |P_{p_u}| + d(p_u,p_v,F) + |P_{p_v}| \geqslant d(u,v,F).$$

Finally, if $w_H(u,v)$ is computed via Equation (2), then it equals $\widehat{d^{\leqslant L}}(u,v,F)$ or $\widehat{d^{\leqslant L}}(x,p,F) + FT_0(p,y,F)$ for some $p \in B_2$ and $y \in \{u,v\}\backslash\{x\}$. The former is not smaller than $d(u,v,F)$ again using Part *(i)*; the latter is at least $d(x,p,F)+d(p,y,F) \geqslant d(u,v,F)$ by the triangle inequality. □

Lemma 21 shows that certain structured paths survive until the output node of an FT-tree, even though full segments are failed in each parent-child transition. This key property lends some importance to the class of $(2f+1)$-decomposable paths that are far away from all failures in F, that is, whose trapezoid $\mathrm{tr}^{\varepsilon/6}_{G-F}(P)$ does not contain any endpoint of an edge in F (Definition 15). We use it to derive upper bounds on the return values $FT_0(u,v,F)$ and $FT_\lambda(u,v,F)$. For this, we need to expand the definition of the decomposable distance $d^{(2f+1)}(u,v,F)$. We define $d^{(2f+1)}_{\varepsilon/6}(u,v,F)$ to be the length of the shortest u-v-path that is $(2f+1)$-decomposable *and far away from* F; or $d^{(2f+1)}_{\varepsilon/9}(u,v,F) = +\infty$ if no such path exists. Note that $d^{(2f+1)}_{\varepsilon/6}(u,v,F) \geqslant d^{(2f+1)}(u,v,F) \geqslant d(u,v,F)$. Further, recall that $D > 0$ is the universal constant used in the output value of FT-trees.

Lemma 23. *Let $u,v \in V$ be two vertices and $F \subseteq E$ a set of $|F| \leqslant f$ edges.*

(i) Let $p_u \in \mathrm{ball}_{G-F}(u,\lambda/2) \cap B_1$ and $p_v \in \mathrm{ball}_{G-F}(v,\lambda/2) \cap B_1$ be pivots of the first type. Then, $FT_\lambda(p_u,p_v,F) + \lambda \leqslant \alpha \cdot d^{(2f+1)}_{\varepsilon/6}(u,v,F) + \alpha\lambda + Df\frac{\log_2(n)}{\varepsilon} \cdot \beta$.

(ii) It holds that $FT_0(u,v,F) \leqslant \alpha \cdot d^{(2f+1)}_{\varepsilon/6}(u,v,F) + Df\frac{\log_2(n)}{\varepsilon} \cdot \beta$.

Similarly to Lemma 21, is important to note that the left-hand side of the inequality in Lemma 23 *(i)* is in terms of the pair (p_u, p_v) while the right-hand side is in terms of (u, v).

Proof of Lemma 23. Let P be the shortest $(2f+1)$-decomposable u-v-path in $G-F$ that is far away from F, that is, $|P| = d^{(2f+1)}_{\varepsilon/6}(u, v, F)$. If no such path exists, the lemma holds vacuously.

We first prove Part *(ii)*. Let ν' the node of $FT_0(u, v)$ that produces the output when the tree is queried with F. By Lemma 21 *(ii)*, P only uses edges in $E \setminus A_{\nu'}$. Since P is $(2+1)$-decomposable, it is also an $(2f+1)$-expath. By definition, $P_{\nu'}$ is the *shortest* $(2f+1)$-expath without granularity from u to v in $G - A_{\nu'}$, whence $|P_{\nu'}| \leqslant |P|$. The query algorithm of $FT_0(u, v)$ returns either $|P_{\nu'}|$ or $\alpha \cdot |P_{\nu'}| + Df\frac{\log_2(n)}{\varepsilon} \cdot \beta$ depending on whether ν' is a leaf. Either way the return value is bounded by $\alpha \cdot |P| + Df\frac{\log_2(n)}{\varepsilon} \cdot \beta = \alpha \cdot d^{(2f+1)}_{\varepsilon/6}(u, v, F) + Df\frac{\log_2(n)}{\varepsilon} \cdot \beta$.

For Part *(i)*, let ν be the output-node of $FT_\lambda(p_u, p_v)$ with query F. The argument is similar as before, but we have to account for the different endpoints of P_ν and P. Consider the shortest (replacement) paths $P(p_u, u, F)$ and $P(v, p_v, F)$ in $G-F$. By Lemma 21 *(i)*, the concatenation $Q = P(p_u, u, F) \circ P \circ P(v, p_v, F)$ lies in $G - A_\nu$. Since $p_u \in \text{ball}_{G-F}(u, \lambda/2) \cap B_1$ and $p_v \in \text{ball}_{G-F}(u, \lambda/2) \cap B_1$, the paths $P(p_u, u, F)$ and $P(v, p_v, F)$ have at most $\lambda/2$ edges. Together with P being $(2f+1)$-decomposable, this shows that Q is an $(2f+1)$-expath with granularity λ from p_u to p_v. It follows that $|P_\nu| \leqslant |Q|$ and

$$\begin{aligned} FT_\lambda(p_u, p_v, F) + \lambda &\leqslant \alpha \cdot |Q| + Df\frac{\log_2(n)}{\varepsilon} \cdot \beta \leqslant \alpha \cdot (|P| + \lambda) + Df\frac{\log_2(n)}{\varepsilon} \cdot \beta \\ &= \alpha \cdot d^{(2f+1)}_{\varepsilon/6}(u, v, F) + \alpha \cdot \lambda + Df\frac{\log_2(n)}{\varepsilon} \cdot \beta. \quad \square \end{aligned}$$

We now turn the above bounds on the return values of the FT-trees into bounds on the edge weight $w_H(u, v)$ in H depending on the existence of certain u-v-paths in $G - F$. First, note that if there is a hop-short path from u to v in $G - F$, then also the replacement path $P(u, v, F)$ has at most L edges. It is easy to see that in this case $w_H(u, v) \leqslant \alpha \cdot d(u, v, F) + \beta$ since $\widehat{d^{\leqslant L}}(u, v, F)$ is part of the minimization in both Equations (1) and (2) and the underlying DSO for hop-short distances has stretch (α, β). We are thus concerned with hop-long u-v-paths. We starting with the "dense ball" case. It is an easy corollary of Lemma 23 *(i)*.

Corollary 24. *Define $\delta = \lambda/L$. Let $F \subseteq E$, $|F| \leqslant f$, be a set of edges and $u, v \in V(F) \cup \{s, t\}$ vertices such that $w_H(u, v)$ is computed using Equation (1). Suppose there exists an u-v-path P in $G - F$ that is the concatenation of at most two replacement paths, hop-long, and far away from all failures in F. Then, with high probability it holds that $w_H(u, v) \leqslant \alpha(1+\delta) \cdot |P| + Df\frac{\log_2(n)}{\varepsilon} \cdot \beta$.*

Proof. Replacement paths are f-decomposable. Therefore, the concatenation P of at most two replacement paths is $(2f+1)$-decomposable. Since P is also assumed to be far away from F, we get $|P| \geqslant d^{(2f+1)}_{\varepsilon/6}(u, v, F)$. Moreover, the path is hop-long, that is, $|P| \geqslant L$.

There exists $p_u \in \text{ball}_{G-F}(u, \lambda/2) \cap B_1$ and $p_v \in \text{ball}_{G-F}(u, \lambda/2) \cap B_1$ such that $w_H(u, v) \leqslant FT_\lambda(p_u, p_v, F) + \lambda$ since the weight is computed via Equation (1). Combining Lemma 23 *(i)* with the lower bounds on $|P|$ gives

$$\begin{aligned} w_H(u, v) \leqslant \alpha \cdot d^{(2f+1)}_{\varepsilon/6}(u, v, F) + \alpha \cdot \lambda + Df\frac{\log_2(n)}{\varepsilon}\beta &\leqslant \alpha \cdot |P| + \alpha \cdot \delta L + Df\frac{\log_2(n)}{\varepsilon}\beta \\ &\leqslant \alpha(1+\delta) \cdot |P| + Df\frac{\log_2(n)}{\varepsilon}\beta. \quad \square \end{aligned}$$

Lemma 25. *Let $F \subseteq E$, $|F| \leqslant f$, be a set of edges and $u, v \in V(F) \cup \{s,t\}$ vertices such that $w_H(u,v)$ is computed using Equation (2). Suppose there exists an u-v-path P in $G - F$ that is the concatenation of at most two replacement paths, hop-long, and far away from all failures in F. Then, with high probability, it holds that $w_H(u,v) \leqslant \alpha \cdot |P| + (Df\frac{\log_2(n)}{\varepsilon}+1) \cdot \beta$.*

Proof. Let $x \in \{u,v\}$ be a vertex for which the pivot tree did not return a sufficiently close pivot of the first type when processing set F. Let $F' \subseteq F$ be the collection of edges corresponding to the parent-child traversals of that pivot tree. With high probability, $\text{ball}_{G-F'}(x, \lambda/2)$ is sparse and the algorithm gains access to the set $\text{ball}_{G-F'}(x,\lambda/2) \cap B_2$ of $\widetilde{O}(fL^f/\lambda)$ pivots of the *second type*. Let $y \in \{u,v\}\backslash\{x\}$ be the other vertex. Equation (2) implies that

$$w_H(u,v) \leqslant \min_{p \in \text{ball}\, G-F'(x,\lambda/2)\cap B_2} \widehat{d^{\leqslant L}}(x,p,F) + FT_0(p,y,F).$$

Recall the covering properties of the set B_2 (below Definition 15). Since P is the concatenation of at most two replacement paths, hop-long, and far away from F, it has a pivot of the second type within distance $\lambda/2$ of either endpoint w.h.p. Let $p^* \in B_2$ be the one around x. The set $\text{ball}_{G-\boldsymbol{F}}(x,\lambda/2)$, is contained in $\text{ball}_{G-\boldsymbol{F'}}(x,\lambda/2)$. That means, the pivot p^* is considered when computing $w_H(u,v)$, giving

$$w_H(u,v) \leqslant \widehat{d^{\leqslant L}}(x,p^*,F) + FT_0(p^*,y,F).$$

The prefix $P[x..p^*]$ has at most $\lambda/2 \leqslant L$ edges, that means, the first term $\widehat{d^{\leqslant L}}(x,p^*,F)$ approximates its length with stretch (α,β). The suffix observes $|P[p^*..y]| \geqslant d^{(2f+1)}_{\varepsilon/6}(p^*,y,F)$. As a subpath of P, the suffix itself is the concatenation of at most two replacement paths, hence $(2f+1)$-decomposable, and far away from F. Using Lemma 23 *(ii)*, we get that

$$\begin{aligned}
\widehat{d^{\leqslant L}}(x,p^*,F) + FT_0(p^*,y,F) &\leqslant \alpha \cdot |P[x..p^*]| + \beta + \alpha \cdot d^{(2f+1)}_{\varepsilon/6}(p^*,y,F) + Df\frac{\log_2(n)}{\varepsilon} \cdot \beta \\
&\leqslant \alpha\Big(|P[x..p^*]| + |P[p^*..y]|\Big) + \left(Df\frac{\log_2(n)}{\varepsilon}+1\right)\cdot\beta \\
&= \alpha \cdot |P| + \left(Df\frac{\log_2(n)}{\varepsilon}+1\right)\cdot\beta. \qquad \square
\end{aligned}$$

We have collected all the lemmas for the different cases in the proof of the stretch below (Lemma 27). In order to tie them together, we use an result by Chechik et al. [CCFK17, Lemma 2.6] about paths P that are *not* for away from F. It states the existence of a detour between the endpoints of P that is not much longer but enjoys the property of being far away from all failures. Namely, one follows P until a certain vertex y. One then leaves the path to reach an endpoint z of some failing edge inside of $\text{tr}^{\varepsilon/6}_{G-F}(P)$. The detour can then be completed by going from z to the other end of P.

Lemma 26 ([CCFK17])**.** *Let $F \subseteq E$, $|F| \leqslant f$, be a set of edges, $u, v \in V(F) \cup \{s,t\}$ vertices, and $P = P(u,v,F)$ their replacement path. If $\text{tr}^{\varepsilon/6}_{G-F}(P) \cap V(F) \neq \emptyset$, then there are vertices $x \in \{u,v\}$, $y \in V(P)$, and $z \in \text{tr}^{\varepsilon/6}_{G-F}(P) \cap V(F)$ with the following properties.*

(i) $|P[x..y]| \leqslant |P|/2$;

(ii) $d(y,z,F) \leqslant \frac{\varepsilon}{6} \cdot d(x,y,F)$;

(iii) $\text{tr}^{\varepsilon/6}_{G-F}\Big(P[x..y] \circ P(y,z,F)\Big) \cap V(F) = \emptyset$.

Thus, the path $P[x..y] \circ P(y,z,F)$ is far away from F and has length at most $(1+\frac{\varepsilon}{6}) \cdot d(x,y,F)$.

Finally, we prove the $(\alpha(1+\varepsilon), \beta)$ stretch of our f-DSO. This is the main result of this section. Recall that the auxiliary graph H has vertex set $V(H) = V(F) \cup \{s,t\}$ and weights on its edges. The output of the DSO is $d_H(s,t)$. At the heart of the proof of Lemma 27 is an induction over the pairwise distances in H. Extra care must be taken to control the additive stretch that accumulates over this induction. The next lemma also completes the proof of the randomized part of Theorem 7. The space, query time, and preprocessing time above where all proven under the assumptions of $\lambda = \Theta(\varepsilon L)$ and $\lambda \leqslant L$. We now need to make the leading constant precise and set $\lambda = \varepsilon L/9$.

Lemma 27. *Define $\lambda = \varepsilon L/9$. Let (α, β) be the stretch of the input f-DSO for distances at most L, where $\beta = o(\frac{\varepsilon^2 L}{f^3 \log n})$. Then, w.h.p. $d(s,t,F) \leqslant d_H(s,t) \leqslant \alpha(1+\varepsilon) \cdot d(s,t,F) + \beta$.*

Proof. We first cover the easy cases. No edge weight $w_H(u,v)$ for $u,v \in V(F) \cup \{s,t\}$ underestimates $d(u,v,F)$ by Lemma 22 *(ii)*. Therefore, the graph distance $d_H(s,t)$ is not smaller than the replacement distance $d(s,t,F)$. If $P(s,t,F)$ has at most L edges, the upper bound is immediate as

$$d_H(s,t) \leqslant w_H(s,t) \leqslant \widehat{d^{\leqslant L}}(s,t,F) \leqslant \alpha \cdot d(s,t,F) + \beta,$$

independently of how the weight is computed. This is a better stretch than what we claimed, but only holds under the assumption that $P(s,t,F)$ is hop-short. The remainder of the proof consists of showing that for hop-long replacement paths we have

$$d_H(s,t) \leqslant \alpha(1+\varepsilon) \cdot d(s,t,F). \tag{3}$$

While it may look as if Inequality (3) is independent of β, it will be proven by subsuming all the additive stretch in the additional $1+\varepsilon$ factor, using that $P(s,t,F)$ is hop-long.

Let $X = Df\frac{\log_2(n)}{\varepsilon}$. Consider all sets $\{u,v\}$ with one or two vertices of H, that is, we include the case $u = v$. ordered ascendingly by the true replacement distance $d(u,v,F)$ (ties are broken arbitrarily). We use $\operatorname{rank}(u,v)$ to denote the rank of the set $\{u,v\}$ in that order. Note the rank is symmetric as H is undirected, that is, $\operatorname{rank}(u,v) = \operatorname{rank}(v,u)$. We claim that, for *all* $\{u,v\}$,

$$d_H(u,v) \leqslant \alpha\left(1+\frac{2}{3}\varepsilon\right) \cdot d(u,v,F) + \operatorname{rank}(u,v)(X+1) \cdot \beta. \tag{4}$$

We prove this by induction over the order. It is clear for all singleton sets where $u = v$. Assume that the assertion holds for all sets coming before $\{u,v\}$. There are three major cases depending on whether $P(u,v,F)$ is hop-short or hop-long and whether is far away from all failures in F. Only Case 3 makes actual use of the induction hypothesis, the others are proven directly.

Case 1. *The replacement path $P(u,v,F)$ is hop-short.*

We have $d_H(u,v) \leqslant \widehat{d^{\leqslant L}}(u,v,F) \leqslant \alpha \cdot d(u,v,F) + \beta \leqslant \alpha(1+\frac{2}{3}\varepsilon)d(u,v,F) + \operatorname{rank}(u,v)(X+1)\beta$.

Case 2. *The replacement path $P(u,v,F)$ is hop-long and far away from F.*

There are two subcases. First, suppose that $w_H(u,v)$ is computed via Equation (1). Applying Corollary 24 with $\delta = \lambda/L = \varepsilon/9$ gives $d_H(u,v) \leqslant w_H(u,v) \leqslant \alpha(1+\frac{\varepsilon}{9}) \cdot d(u,v,F) + X\beta$ w.h.p. Otherwise, the weight is computed via Equation (2). Lemma 25 shows that $d_H(u,v) \leqslant w_H(u,v) \leqslant \alpha \cdot d(u,v,F) + (X+1) \cdot \beta$ w.h.p. Both bounds are never larger than what we claim in Inequality (4).

Case 3. *The replacement path $P(u,v,F)$ is hop-long but close to F.*

Let $P = P(u, v, F)$. By Lemma 26, there exists an endpoint $x \in \{u, v\}$ of P, some other vertex $y \in V(P)$ on the path, and and an endpoint of a failing edge $z \in \mathrm{tr}^{\varepsilon/6}_{G-F}(P) \cap V(F)$ in the trapezoid such that the concatenation $Q = P[x..y] \circ P(y, z, F)$ is far away from all failures in F. $P(y, z, F)$ denotes the replacement path from y to z. Further note that $P[x..y]$ is the replacement path from x to y, since it is a subpath of $P(u, v, F)$. In summary Q is the concatenation of two replacement paths and far away from F. Lemma 26 also gives $|P(y, z, F)| = d(y, \boldsymbol{z}, F) \leqslant \frac{\varepsilon}{6} d(x, \boldsymbol{y}, F)$. So Q is only slightly larger than $|P[x..y]| = d(x, y, F)$, namely, of length $(1+\frac{\varepsilon}{6})\, d(x, y, F)$. For notational convenience, let w be the other endpoint of P in $\{u, v\} \setminus \{x\}$.

On our way to establish Inequality (4) also in this case, we claim the following bound on the weight of the edge $\{x, z\}$ in H,

$$w_H(x, z) \leqslant \alpha \left(1 + \frac{\varepsilon}{9}\right)\left(1 + \frac{\varepsilon}{6}\right) \cdot d(x, y, F) + (X{+}1) \cdot \beta. \tag{5}$$

Note that the left-hand side is in terms of the pair (x, z) while the right-hand side is in terms of (x, y). Translating between the two is done via $d(x, y, F) \leqslant (1 + \frac{\varepsilon}{6})\, d(x, y, F)$ by the choice of vertex z. Intuitively, Inequality (5) states that following Q from x to z and continuing to the other endpoint w from there is not much more expensive than going directly from x to w along P.

We distinguish the same three cases as above but now with respect to path Q. In the first one, Q is hop-short. As in Case 1, we get

$$w_H(x, z) \leqslant d^{\leqslant L}(x, z, F) \leqslant \alpha \cdot d(x, \boldsymbol{z}, F) + \beta \leqslant \alpha \left(1 + \frac{\varepsilon}{6}\right) \cdot d(x, \boldsymbol{y}, F) + \beta.$$

Now assume Q is hop-long and $w_H(x, z)$ is computed using Equation (1). This is handled by Corollary 24 which also applies to Q as it is the concatenation of two replacement paths and far away from F. Reecall that $\delta = \varepsilon/9$. We get

$$\begin{aligned} w_H(x, z) &\leqslant \alpha\,(1{+}\delta) \cdot d(x, \boldsymbol{z}, F) + (X{+}1) \cdot \beta \\ &\leqslant \alpha \left(1 + \frac{\varepsilon}{9}\right)\left(1 + \frac{\varepsilon}{6}\right) \cdot d(x, \boldsymbol{y}, F) + (X{+}1) \cdot \beta. \end{aligned}$$

Finally, assume $w_H(x, z)$ is computed using Equation (2). From Lemma 25, we get $w_H(x, z) \leqslant \alpha \cdot |Q| + (X{+}1) \cdot \beta \leqslant \alpha(1 + \frac{\varepsilon}{6}) \cdot d(x, y, F) + (X{+}1) \cdot \beta$, which implies Inequality (5).

Now that we have a bound on the weight $w_H(x, z)$, we prove Inequality (4) in Case 3. Recall that this case assumes that $P = P(u, v, F)$ is hop-long but *close* to some failure in F. Also recall that $\{x, w\} = \{u, v\}$ is the same pair of vertices. The other vertex z lies in the $\frac{\varepsilon}{6}$-trapezoid of P. Since $\frac{\varepsilon}{6} \leqslant \frac{12}{7 \cdot 6} < 1$, *every* element of $\mathrm{tr}^{\varepsilon/6}_{G-F}(P)$ is closer to w in $G - F$ than x is. In particular, we have $d(z, w, F) < d(x, w, F) = d(u, v, F)$, whence $\mathrm{rank}(z, w)$ is strictly smaller than $\mathrm{rank}(x, w) = \mathrm{rank}(u, v)$. We apply the induction hypothesis to $\{z, w\}$. It states that $d_H(z, w) \leqslant \alpha(1 + \frac{2}{3}\varepsilon) \cdot d(z, w, F) + \mathrm{rank}(z, w)(X + 1) \cdot \beta$. Inequality (5) gives

$$\begin{aligned} d_H(u, v) = d_H(x, w) &\leqslant w_H(x, z) + d_H(z, w) \\ &\leqslant \alpha\left(1 + \frac{\varepsilon}{9}\right)\left(1 + \frac{\varepsilon}{6}\right) d(x, y, F) + (X{+}1)\,\beta + \alpha\left(1 + \frac{2}{3}\varepsilon\right) d(z, w, F) + \mathrm{rank}(z, w)(X{+}1)\,\beta. \end{aligned}$$

We first aggregate the multiples of β and then those of α. We have $(\mathrm{rank}(z, w) + 1)(X{+}1) \cdot \beta \leqslant \mathrm{rank}(x, w)(X{+}1) \cdot \beta = \mathrm{rank}(u, v)(X{+}1) \cdot \beta$ For the other terms, we use that $\varepsilon \leqslant 12/7$ implies

$$\left(1 + \frac{\varepsilon}{9}\right)\left(1 + \frac{\varepsilon}{6}\right) + \left(1 + \frac{2}{3}\varepsilon\right)\frac{\varepsilon}{6} = \frac{7}{54}\varepsilon^2 + \frac{4}{9}\varepsilon + 1 \leqslant 1 + \frac{2}{3}\varepsilon.$$

We omit the factor α in the following calculation.

$$\begin{aligned}
&\left(1+\frac{\varepsilon}{9}\right)\left(1+\frac{\varepsilon}{6}\right)d(x,y,F)+\left(1+\frac{2}{3}\varepsilon\right)d(z,w,F)\\
&\quad\leqslant\left(1+\frac{\varepsilon}{9}\right)\left(1+\frac{\varepsilon}{6}\right)d(x,y,F)+\left(1+\frac{2}{3}\varepsilon\right)\Big(d(z,y,F)+d(y,w,F)\Big)\\
&\quad\leqslant\left(1+\frac{\varepsilon}{9}\right)\left(1+\frac{\varepsilon}{6}\right)d(x,y,F)+\left(1+\frac{2}{3}\varepsilon\right)\frac{\varepsilon}{6}d(x,y,F)+\left(1+\frac{2}{3}\varepsilon\right)d(y,w,F)\\
&\quad=\left(\left(1+\frac{\varepsilon}{9}\right)\left(1+\frac{\varepsilon}{6}\right)+\left(1+\frac{2}{3}\varepsilon\right)\frac{\varepsilon}{6}\right)d(x,y,F)+\left(1+\frac{2}{3}\varepsilon\right)d(y,w,F)\\
&\quad\leqslant\left(1+\frac{2}{3}\varepsilon\right)d(x,y,F)+\left(1+\frac{2}{3}\varepsilon\right)d(y,w,F)=\left(1+\frac{2}{3}\varepsilon\right)d(x,w,F)=\left(1+\frac{2}{3}\varepsilon\right)d(u,v,F).
\end{aligned}$$

Recombining the terms depending on α and β completes Case 3 and establishes Inequality (4).

We now use this to also prove Inequality (3), which states that $d_H(s,t)\leqslant\alpha(1+\varepsilon)\cdot d(s,t,F)$ is true for those queries for which $P(s,t,F)$ is hop-long. This is the last part of the proof that is still open. Inequality (4) applied to the pair $\{s,t\}$ states that

$$d_H(s,t)\leqslant\alpha\left(1+\frac{2}{3}\varepsilon\right)\cdot d(s,t,F)+\operatorname{rank}(s,t)(X+1)\cdot\beta.$$

It is thus enough to prove $\operatorname{rank}(s,t)(X+1)\beta\leqslant\alpha\frac{\varepsilon}{3}d(s,t,F)$.

Observe that the replacement path being hop-long implies that $d(s,t,F)\geqslant L$. There are $\binom{|V(F)\cup\{s,t\}|}{2}+|V(F)\cup\{s,t\}|\leqslant 2f^2+5f+3\leqslant 6f^2$ sets $\{u,v\}$ in H, which is an upper bound on $\operatorname{rank}(s,t)$. The last estimate uses $f\geqslant 2$. The restriction $\beta=o(\frac{\varepsilon^2 L}{f^3\log n})$ implies $7Df^3\frac{\log_2(n)}{\varepsilon}\beta\leqslant\frac{\varepsilon}{3}L$ for all sufficiently large n. Together with $\alpha\geqslant 1$, we get

$$\begin{aligned}
\operatorname{rank}(s,t)(X+1)\,\beta&\leqslant 6f^2\left(Df\frac{\log_2(n)}{\varepsilon}+1\right)\beta\leqslant 7Df^3\frac{\log_2(n)}{\varepsilon}\beta\\
&\leqslant\frac{\varepsilon}{3}L\leqslant\alpha\frac{\varepsilon}{3}L\leqslant\alpha\frac{\varepsilon}{3}d(s,t,F).\quad\square
\end{aligned}$$

7 Derandomization

We now derandomize the steps leading up to Theorem 5. The process of making distance oracles fault-tolerant for short distances (Theorem 6) is already deterministic. We are thus left with the distance oracle in Theorem 1 and the pivots in Theorem 7. We use the critical-path framework of Alon, Chechik and Cohen [ACC19]. This means computing a deterministic hitting set for the namesake *critical (replacement) paths* and use it in the construction of the oracle. The crucial issue is to make the set of paths as small as possible and easily identifiable in order to keep the derandomization efficient. Once assembled, the paths are given as input to the straightforward greedy algorithm that obtains the hitting set by iteratively selecting the vertex contained in the most yet unhit paths. We treat the paths are mere sets of vertices. This allows us to seamlessly extend the approach to the sets $\operatorname{ball}_{G-F}(u,\lambda/2)$ when derandomizing Theorem 7.

The following folklore lemma states the properties of the greedy algorithm. A formal proof can be found in [ACC19]; King [Kin99] gave an alternative algorithm with similar parameters.

Lemma 28. *Let V be a set of $|V|=n$ vertices, and $M\leqslant n$ and q be positive integers (possibly depending on n). Let $S_1,\dots,S_q\subseteq V$ be sets of vertices that, for all indices $1\leqslant k\leqslant q$, satisfy $|S_k|=\Omega(M)$. The* GreedyPivotSelection *algorithm runs in time $\widetilde{O}(qM)$ and outputs a set $B\subseteq V$ of cardinality $|B|=O(n\log(q)/M)$ such that, for every k, it holds that $B\cap S_k\neq\emptyset$.*

Preferably, the time needed for the derandomization does not increase the asymptotic bounds of Theorems 1 and 7. We thus need to find the critical vertex sets as quickly as possible while also keeping their number q small. If, however, we choose the minimum size M too small, we end up with too many pivots.

7.1 Derandomizing the Near-Additive Distance Oracle

The derandomization of the distance oracles is a good introduction how we employ Lemma 28. We focus the discussion on Theorem 1. The same ideas also apply to Theorem 3. The only random choice in the construction is which elements to include in the set B of pivots. Recall that the input graph G is assumed to be weighted. For any non-negative radius r, the set V_r contains those vertices u that have at most K vertices within (weighted) distance r around u. Here, K is the parameter from Theorem 1. The only time we use a property of B other than its size is right before Lemma 9. We require that any vertex $v \in V \setminus V_r$ has a pivot $p(v) \in B$ with $d(v, p(v)) \leqslant r$.

Let $K[v]$ be the set of the K closest vertices to v. If v is connected to fewer than K vertices, then $K[v]$ is the entire component. We claim that it is enough to find a hitting set for all the $K[v]$ that have exactly K elements. To see this, first note that disconnected vertices, i.e., $r = +\infty$, are handled directly by the oracle. Suppose $v \in V \setminus V_r$ for some finite r. The set $\text{ball}(v, r)$ has more than K vertices. On the one hand, this gives $|K[v]| = K$ as more than K vertices are connected to v. On the other hand, the closest K among them all have distance at most r. Hitting either one of them is sufficient.

In terms of the parameters of Lemma 28, there are $q \leqslant n$ critical sets that we need to hit. The lower bound on their size translates to choosing $M = K$. We thus get a hitting set B of size $O(n \log(q)/M) = \widetilde{O}(n/K)$, the same as we had for Theorem 1. The sets $K[v]$ are computed anyway during the preprocessing so this incurs no extra charge. GreedyPivotSelection runs in time $\widetilde{O}(qM) = \widetilde{O}(nK) = \widetilde{O}(n^{3/2})$ which is dominated by the APSP computation in the preprocessing.

7.2 Derandomizing the General Distance Sensitivity Oracle

We now turn to Theorem 7. In particular, we restrict our attention to unweighted graphs. The only source of randomness are the pivot sets B_1 and B_2. We recall their covering properties. The pivots of the first type in B_1 are used to hit all the sets $\text{ball}_{G-F}(u, \lambda/2)$ provided that they are dense, that is, contain more than L^f vertices. The set B_2 hits the (unique) shortest paths in G on at least L edges, as well as the first and last $\lambda/2$ vertices of all hop-long concatenations of up to two replacement paths in any $G-F$. In the construction of the oracle, we only used pivots of the second type on concatenations that are far away from F. However, we do not use this information for the derandomization.

The set B_2 is much easier to make deterministic than B_1, which is why we start with the pivots of the second type. Recall that we compute all-pairs shortest paths in G during preprocessing (to get expaths in additional time $\widetilde{O}(fm)$). While doing so, we collect all paths that have length at least $M_2 = \lambda/(4f+2)$. Evidently, there are at most $q_2 = O(n^2)$ such paths. Lemma 28 with parameters M_2 and q_2 gives a set B_2 with $\widetilde{O}(fn/\lambda)$ pivots. This is the same number that was guaranteed with high probability for the random construction, and used throughout Section 6. The running time of GreedyPivotSelection is $\widetilde{O}(n^2\lambda/f)$.

To show that it is enough to hit all shortest paths in G of length $\lambda/(4f+2)$, we use that concatenations of at most two replacement paths are $(2f+1)$-decomposable [ABK+02]. Consider such a concatenation with at least λ edges in total, and let P be its prefix (or suffix) with $\lambda/2$ edges. P itself is $(2f+1)$-decomposable, that is, a concatenation of at most $2f+1$ shortest paths in

G. One of them must have at least $\lambda/(4f+2)$ edges. The same argument shows that B_2 intersects all replacement paths with at least $\lambda/2$ edges. The hop-long shortest path in G are hit by design.

If we were to use the same approach for the pivots of the first type, we would get way too many vertices. Conversely, brute-forcing the dense balls in the graphs $G-F$ for *all* the failure sets $F \subseteq E$ with at most $|F| \leqslant f$ edges takes too much time. Instead, we use the pivot trees introduced in Section 6.1 also for the derandomization. Recall that we build a pivot tree with at most $f+1$ levels for each vertex $u \in V$. Each node stores a path of at most $\lambda/2$ edges, or a set of $\widetilde{O}(fL^f/\lambda)$ pivots of the second type in case $\mathrm{ball}_{G-F'}(u, \lambda/2)$ is sparse. The set F' corresponds to the edges that are associated with the parent-child traversals on the path from the root to the current node.

We interleave the level-wise construction of the pivot trees with derandomization phases. Consider all the sets $\mathrm{ball}_{G-F'}(u, \lambda/2)$ that were computed to create the nodes of the current level of all trees. In the subsequent derandomization phase, it is not enough to hit only the dense balls among them. The reason is as follows. Whenever there is no pivot of the first type in the vicinity of u, we store all pivots of the second type that are close to u. The greedily selected pivots in B_2 may not be as evenly spread as the random ones, potentially resulting in even a sparse ball receiving much more than $\widetilde{O}(fL^f/\lambda)$ many pivots of the second type. We must therefore hit all balls that have too large of an intersection with B_2.

In the first level, we construct the root nodes of the pivot trees for all $u \in V$ by computing the sets $\mathrm{ball}_G(u, \lambda/2)$ in the original graph G via breath-first searches. We use a slightly different stopping criteria than the one in Section 6.1. The BFS halts once the whole ball is explored or L^f vertices have been visited (whatever happens first). This way, we can keep the computing time per ball at $O(L^{2f})$. To apply Lemma 28, let $S_1, \ldots, S_{q_1}$ be all balls whose computation was stopped at L^f-vertices or which have an intersection with B_2 of size more than fL^f/λ. This means that each of them has at least $M_1 = fL^f/\lambda$ elements. We get a hitting set of $O(n \log(q_1)\lambda/(fL^f))$ vertices. For each $u \in V$, if $\mathrm{ball}_G(u, \lambda/2)$ is among the sets S_k, we store the path from u to the closest selected element. Otherwise, we store $\mathrm{ball}_G(u, \lambda/2) \cap B_2$, which has size $O(fL^f/\lambda)$. In the latter case, we mark the node as a leaf. In each subsequent level up to height f, we iterate this process in that for each non-leaf node, we create a child for each edge on the stored path, and compute the ball around u in the corresponding graph $G-F'$. In the first level, there are at most $O(n)$ balls given to GreedyPivotSelection, but this can grow up to $q_1 = O(n\lambda^f)$ in the last level.

As the new set of pivots of the first type B_1, we take the union of all the pivots generated in the derandomization phases. These are $O(f \cdot n \log(n\lambda^f) \cdot \frac{\lambda}{fL^f}) = \widetilde{O}(fn\lambda/L^f)$ and thus a factor $O(\lambda)$ larger than what we had using randomization. We prove that this does not increase the space and preprocessing time by more than constant factors.

We redo the calculations in the paragraph "Finishing the preprocessing" (just before Section 6.2) with the new value of $|B_1|$. We use the assumptions $\lambda \leqslant L$ and $f \leqslant L$ for simplifications.

$$
\begin{aligned}
&|B_1|^2 \cdot \left(\lambda + O\left(f\frac{\log n}{\varepsilon}\right)\right)\left(O(\lambda^f) + O\left(\frac{\log n}{\varepsilon}\right)^f\right) \\
&= \widetilde{O}\left(f^2\frac{n^2\lambda^{f+3}}{L^{2f}}\right) + \widetilde{O}\left(f^2\frac{n^2\lambda^3}{L^{2f}}\right)O\left(\frac{\log n}{\varepsilon}\right)^f + \widetilde{O}\left(f^3\frac{n^2\lambda^{f+2}}{L^{2f}}\right)O\left(\frac{\log n}{\varepsilon}\right) + \widetilde{O}\left(f^3\frac{n^2\lambda^2}{L^{2f}}\right)O\left(\frac{\log n}{\varepsilon}\right)^{f+1} \\
&= \widetilde{O}\left(f^4\frac{n^2}{L^{f-3}}\right) + \widetilde{O}\left(f^2\frac{n^2}{L^{2f-3}}\right)O\left(\frac{\log n}{\varepsilon}\right)^f + \widetilde{O}\left(f^2\frac{n^2}{L^{f-3}}\right)O\left(\frac{\log n}{\varepsilon}\right) + \widetilde{O}\left(f^2\frac{n^2}{L^{2f-3}}\right)O\left(\frac{\log n}{\varepsilon}\right)^{f+1}.
\end{aligned}
$$

The space of the FT-trees without granularity between the pairs in $B_2 \times V$ stays the same. For the derandomization, we assume $f \geqslant 4$, giving $n^2/L^{f-3} \leqslant n^2/L$. All terms (for both kinds of FT-trees) are thus at most $\widetilde{O}(f^2n^2/L) \cdot O(\log(n)/\varepsilon)^{f+2}$. This is the same as in the randomized construction.

The term in the preprocessing time that depends on the new size of B_1 is

$$|B_1|^2 \cdot \left(O(\lambda^f) + O\left(\frac{\log n}{\varepsilon}\right)^f\right) = \widetilde{O}\left(f^2\frac{n^2\lambda^{f+2}}{L^{2f}}\right) + \widetilde{O}\left(f^2\frac{n^2\lambda^2}{L^{2f}}\right)O\left(\frac{\log n}{\varepsilon}\right)^f$$
$$= \widetilde{O}\left(f^2\frac{n^2}{L^{f-2}}\right) + \widetilde{O}\left(f^2\frac{n^2}{L^{2f-2}}\right)O\left(\frac{\log n}{\varepsilon}\right)^f = \widetilde{O}\left(f\frac{n^2}{L^{f-3}}\right) + \widetilde{O}\left(f\frac{n^2}{L^{2f-3}}\right)O\left(\frac{\log n}{\varepsilon}\right)^f.$$

The corresponding term for the FT-trees without granularity is $O(fn/\lambda)\cdot O(\log(n)/\varepsilon)^f = O(fn/L)\cdot O(\log(n)/\varepsilon)^{f+1}$. Since $n^2/L^{f-3} \leqslant n^2/L$, this term still dominates this part of the preprocessing.

We still have to account for the time spend on the derandomization itself. Recall that, for B_2, this is $\widetilde{O}(n^2\lambda/f) = \widetilde{O}(n^2\varepsilon L/f)$. Regarding B_1, the time for the $f+1$ derandomization phases can be estimated as being an $O(f)$-factor larger than the one for the last phase. Lemma 28 gives $\widetilde{O}(f\cdot n\lambda^f\cdot\frac{fL^f}{\lambda}) = \widetilde{O}(f^2L^{2f-1}n)$. We combine the times with those needed to precompute the short-distance DSO, the pivot trees, and LCA data structures just like in Section 6.1. Using that $f^2L^{2f-1} < L^{3f}$, we arrive at

$$\mathsf{T}_L + O(L^{3f}n) + \widetilde{O}\left(\frac{n^2\varepsilon L}{f}\right) + \widetilde{O}\left(f^2L^{2f-1}n\right) + \widetilde{O}(fm)\cdot\widetilde{O}\left(f\frac{n^2}{L}\right)\cdot O\left(\frac{\log n}{\varepsilon}\right)^{f+1}$$
$$= \mathsf{T}_L + O(L^{3f}n) + \widetilde{O}\left(\frac{n^2\varepsilon L}{f}\right) + \widetilde{O}\left(f^2\frac{mn^2}{L}\right)\cdot O\left(\frac{\log n}{\varepsilon}\right)^{f+1}.$$

If $L = \widetilde{O}(\sqrt{f^3m/\varepsilon})$, we have $\widetilde{O}(n^2\varepsilon L/f) = \widetilde{O}(f^2mn^2/L)$. Therefore, we get the same asymptotic preprocessing time as with randomization.

8 Proof of Theorem 5

We chain the new distance oracle of Corollary 2 with the reductions in Theorems 6 and 7 to obtain a deterministic f-DSO for unweighted graphs with near-additive stretch and subquadratic space.

Theorem 5. *Let ℓ be a positive integer constant and G be an undirected and unweighted graph with n vertices and m edges and unique shortest paths. For any $0 < \gamma \leqslant (\ell+1)/2$, sensitivity $2 \leqslant f = o(\log(n)/\log\log n)$, and approximation parameter $\varepsilon = \omega(\sqrt{\log(n)}/n^{\frac{\gamma}{2(\ell+1)(f+1)}})$, there exists an f-edge fault-tolerant distance sensitivity oracle for G that has*

- *stretch* $((1+\frac{1}{\ell})(1+\varepsilon), 2)$,
- *space* $n^{2-\frac{\gamma}{(\ell+1)(f+1)}+o(1)}/\varepsilon^{f+2}$,
- *query time* $O(n^\gamma/\varepsilon^2)$,
- *preprocessing time* $n^{2+\gamma+o(1)} + mn^{2-\frac{\gamma}{(\ell+1)(f+1)}+o(1)}/\varepsilon^{f+1}$.

Proof. The static distance oracle of Corollary 2 is instantiated for unweighted graphs, i.e., with maximum edge weight $W = 1$, and parameter $c = \gamma\ell/(\ell+1)$. Since $\gamma \leqslant (\ell+1)/2$, we have $c \leqslant \ell/2$. The DO has stretch $(\alpha,\beta) = (1+\frac{1}{\ell}, 2)$, space $\mathsf{S} = \widetilde{O}(n^{2-\frac{\gamma}{\ell+1}})$, and query time $\mathsf{Q} = O(n^{\gamma t/(1+\ell)})$. The time to compute it is $\mathsf{T} = O(mn)$ when a BFS from every vertex is used for APSP.

We then plug this oracle into Theorem 6 with a cut-off parameter $L = n^{\frac{c}{\ell(f+1)}} = n^{\frac{\gamma}{(\ell+1)(f+1)}}$ that distinguishes between hop-long and hop-short distances. Note that this implies $L^f = n^{(1-\frac{1}{f+1})\frac{\gamma}{\ell+1}}$.

The assumption $f = o(\log(n)/\log\log n)$ implies that $O(f)^f = O(\log n)^f = n^{o(1)}$. We use this for the space of the resulting hop-short distance *sensitivity* oracle

$$\mathsf{S}_L = O(fL\log n)^f \cdot \mathsf{S} = L^f\, n^{2-\frac{\gamma}{\ell+1}+o(1)} = n^{2-\frac{\gamma}{(\ell+1)(f+1)}+o(1)}.$$

Since $\alpha L + \beta + f^2 L = \widetilde{O}(n^{\frac{\gamma}{(\ell+1)(f+1)}})$ is negligible compared to $\mathsf{Q} = O(n^{\gamma\ell/(\ell+1)})$, the query time of the intermediate data structure is $\mathsf{Q}_L = \widetilde{O}(f\mathsf{Q}) = \widetilde{O}(n^{\gamma\ell/(\ell+1)})$. We defer the analysis of the preprocessing time to the end of the proof as it uses similar simplifications as introduced next.

The condition $\varepsilon = \omega(\sqrt{\log(n)}/n^{\frac{\gamma}{2(\ell+1)(f+1)}})$ bounding how fast ε can go to 0 (if any) ensures that $\beta = 2 = o(\frac{\varepsilon^2 L}{f^3 \log n})$. Also, we have $L = n^{\frac{\gamma}{(\ell+1)(f+1)}} = O(n^{1/(2(f+1))}) = \widetilde{O}(\sqrt{f^3 m/\varepsilon})$. We can thus apply the deterministic version of Theorem 7 to the f-DSO for short distances to get a general f-DSO with slightly increased stretch $((1+\frac{1}{\ell})(1+\varepsilon), 2)$.

We first discuss its space. We insert our choice of L into the second and third term of the space bound in Theorem 7.

$$\widetilde{O}(fL^{2f-1}n) = \widetilde{O}\left(n^{\frac{2f-1}{f+1}\frac{\gamma}{\ell+1}} \cdot n\right) = \widetilde{O}\left(n^{1+\frac{2f-1}{f+1}\frac{\gamma}{\ell+1}}\right).$$

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

They have a factor $n^{-1+\frac{2f}{f+1}\frac{\gamma}{\ell+1}+o(1)}/\varepsilon^{f+2}$ between them. Since $\gamma/(\ell+1) \leqslant 1/2 < (f+1)/(2f)$, the latter term dominates. With high probability, the total space of our general DSO is

$$\mathsf{S}_L + \widetilde{O}\left(f^2\frac{n^2}{L}\right) \cdot O\left(\frac{\log n}{\varepsilon}\right)^{f+2} = n^{2-\frac{\gamma}{(\ell+1)(f+1)}+o(1)} + \frac{n^{2-\frac{\gamma}{(\ell+1)(f+1)}+o(1)}}{\varepsilon^{f+2}} = \frac{n^{2-\frac{\gamma}{(\ell+1)(f+1)}+o(1)}}{\varepsilon^{f+2}}.$$

Its query time is $\widetilde{O}(f^5\frac{L^{f-1}}{\varepsilon^2} \cdot (\mathsf{Q}_L + f)) = \widetilde{O}(n^{(1-\frac{2}{f+1})\frac{\gamma}{(\ell+1)}} \cdot n^{\frac{\gamma\ell}{\ell+1}}/\varepsilon^2) = \widetilde{O}(n^\gamma/\varepsilon^2)$.

The only thing left to prove is the preprocessing time. Recall that the static DO is computed in time $\mathsf{T} = O(mn)$. Consider the first term of the preprocessing time in Theorem 6 The $O(fL\log n)^f = n^{(1-\frac{1}{f+1})\frac{\gamma}{(\ell+1)}+o(1)}$ part is negligible compared to $\mathsf{Q} = O(n^{\frac{\gamma\ell}{\ell+1}})$. Also, we have $(\alpha L+\beta)^f = O(L)^f = n^{(1-\frac{1}{f+1})\frac{\gamma}{(\ell+1)}+o(1)}$. The first step of the transformation thus runs in time

$$\begin{aligned}
\mathsf{T}_L &= \widetilde{O}(n^2 \cdot (\alpha L+\beta)^f) \cdot (O(fL\log n)^f + \mathsf{Q}) + O(fL\log n)^f \cdot \mathsf{T} \\
&= n^{2+(1-\frac{1}{f+1})\frac{\gamma}{(\ell+1)}+o(1)} \cdot \mathsf{Q} + n^{(1-\frac{1}{f+1})\frac{\gamma}{(\ell+1)}+o(1)} \cdot \mathsf{T} \\
&= n^{2+(1-\frac{1}{f+1})\frac{\gamma}{(\ell+1)}+\frac{\gamma\ell}{\ell+1}+o(1)} + mn^{1+(1-\frac{1}{f+1})\frac{\gamma}{\ell+1}+o(1)} \\
&= n^{2+\gamma+o(1)} + mn^{1+\frac{\gamma}{\ell+1}+o(1)}.
\end{aligned}$$

The running time of the reduction in Theorem 7 is

$$\mathsf{T}_L + O(L^{3f}n) + \widetilde{O}\left(f^2\frac{mn^2}{L}\right) \cdot O\left(\frac{\log n}{\varepsilon}\right)^{f+1}.$$

The second term is $O(L^{3f}n) = O(n^{1+\frac{3f}{f+1}\frac{\gamma}{\ell+1}})$. With our implicit assumption of $m \geqslant n$, this is dominated by $\widetilde{O}(f^2mn^2/L) \cdot O(\log(n)/\varepsilon)^{f+1} = mn^{2-\frac{1}{f+1}\frac{\gamma}{\ell+1}+o(1)}/\varepsilon^{f+1}$. Therefore, the total

preprocessing time in our case simplifies to

$$\begin{aligned}\mathsf{T}_L + \frac{mn^{2-\frac{1}{f+1}\frac{\gamma}{\ell+1}+o(1)}}{\varepsilon^{f+1}} &= n^{2+\gamma+o(1)} + mn^{1+\frac{\gamma}{\ell+1}+o(1)} + \frac{mn^{2-\frac{\gamma}{(\ell+1)(f+1)}+o(1)}}{\varepsilon^{f+1}} \\ &= n^{2+\gamma+o(1)} + \frac{mn^{2-\frac{\gamma}{(\ell+1)(f+1)}+o(1)}}{\varepsilon^{f+1}}.\end{aligned}$$

□

Acknowledgements

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)". The third author is partially supported by Google India Research Awards.

References

[ABK+02] Yehuda Afek, Anat Bremler-Barr, Haim Kaplan, Edith Cohen, and Michael Merritt. Restoration by Path Concatenation: Fast Recovery of MPLS Paths. *Distributed Computing*, 15:273–283, 2002. doi:10.1007/s00446-002-0080-6.

[ACC19] Noga Alon, Shiri Chechik, and Sarel Cohen. Deterministic Combinatorial Replacement Paths and Distance Sensitivity Oracles. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming, (ICALP)*, pages 12:1–12:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.12.

[AG11] Ittai Abraham and Cyril Gavoille. On Approximate Distance Labels and Routing Schemes with Affine Stretch. In *Proceedings of the 25th International Symposium on Distributed Computing (DISC)*, pages 404–415, 2011. doi:10.1007/978-3-642-24100-0_39.

[AG13] Rachit Agarwal and P. Brighten Godfrey. Distance Oracles for Stretch Less Than 2. In *Proceedings of the 24th Symposium on Discrete Algorithms (SODA)*, pages 526–538, 2013. doi:10.1137/1.9781611973105.38.

[Aga14] Rachit Agarwal. The Space-Stretch-Time Tradeoff in Distance Oracles. In *Proceedings of the 22th Annual European Symposium on Algorithms (ESA)*, pages 49–60, 2014. doi:10.1007/978-3-662-44777-2_5.

[AR20] Maor Akav and Liam Roditty. An Almost 2-Approximation for All-Pairs of Shortest Paths in Subquadratic Time. In *Proceedings of the 14th Symposium on Discrete Algorithms (SODA)*, pages 1–11, 2020. doi:10.1137/1.9781611975994.1.

[BCC+23] Davide Bilò, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, Simon Krogmann, and Martin Schirneck. Compact Distance Oracles with Large Sensitivity and Low Stretch. In *Proceedings of the 18th Algorithms and Data Structures Symposium (WADS)*, pages 149–163, 2023. doi:10.1007/978-3-031-38906-1_11.

[BCC+24] Davide Bilò, Shiri Chechik, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, Simon Krogmann, and Martin Schirneck. Approximate Distance Sensitivity Oracles in Subquadratic Space. *TheoretiCS*, 3:15:1–15:47, 2024. doi:10.46298/theoretics.24.15.

[BCFS21] Davide Bilò, Sarel Cohen, Tobias Friedrich, and Martin Schirneck. Near-Optimal Deterministic Single-Source Distance Sensitivity Oracles. In *Proceedings of the 29th European Symposium on Algorithms (ESA)*, pages 18:1–18:17, 2021. doi:10.4230/LIPIcs.ESA.2021.18.

[BF00] Michael A. Bender and Martín Farach-Colton. The LCA Problem Revisited. In *Proceedings of the 4th Latin American Symposium Theoretical Informatics (LATIN)*, pages 88–94, 2000. doi:10.1007/10719839_9.

[BGS09] Surender Baswana, Vishrut Goyal, and Sandeep Sen. All-pairs Nearly 2-Approximate Shortest Paths in $O(n^2 \,\mathsf{polylog}\, n)$ Time. *Theoretical Computer Science*, 410:84–93, 2009. doi:10.1016/j.tcs.2008.10.018.

[BK08] Aaron Bernstein and David R. Karger. Improved Distance Sensitivity Oracles via Random Sampling. In *Proceedings of the 19th Symposium on Discrete Algorithms (SODA)*, pages 34–43, 2008. URL: https://dl.acm.org/doi/abs/10.5555/1347082.1347087.

[BK09] Aaron Bernstein and David R. Karger. A Nearly Optimal Oracle for Avoiding Failed Vertices and Edges. In *Proceedings of the 41st Symposium on Theory of Computing (STOC)*, pages 101–110, 2009. doi:10.1145/1536414.1536431.

[BK13] Surender Baswana and Neelesh Khanna. Approximate Shortest Paths Avoiding a Failed Vertex: Near Optimal Data Structures for Undirected Unweighted Graphs. *Algorithmica*, 66:18–50, 2013. doi:10.1007/s00453-012-9621-y.

[BS19] Jan van den Brand and Thatchaphol Saranurak. Sensitive Distance and Reachability Oracles for Large Batch Updates. In *Proceedings of the 60th Symposium on Foundations of Computer Science (FOCS)*, pages 424–435, 2019. doi:10.1109/FOCS.2019.00034.

[CC20] Shiri Chechik and Sarel Cohen. Distance Sensitivity Oracles with Subcubic Preprocessing Time and Fast Query Time. In *Proccedings of the 52nd Symposium on Theory of Computing (STOC)*, pages 1375–1388, 2020. doi:10.1145/3357713.3384253.

[CCE13] Sergio Cabello, Erin W. Chambers, and Jeff Erickson. Multiple-Source Shortest Paths in Embedded Graphs. *SIAM Journal on Computing*, 42:1542–1571, 2013. doi:10.1137/120864271.

[CCFK17] Shiri Chechik, Sarel Cohen, Amos Fiat, and Haim Kaplan. $(1+\varepsilon)$-Approximate f-Sensitive Distance Oracles. In *Proceedings of the 28th Symposium on Discrete Algorithms (SODA)*, pages 1479–1496, 2017. doi:10.1137/1.9781611974782.96.

[Che14] Shiri Chechik. Approximate Distance Oracles with Constant Query Time. In *Proceedings of the 46th Symposium on Theory of Computing (STOC)*, pages 654–663, 2014. doi:10.1145/2591796.2591801.

[Che15] Shiri Chechik. Approximate Distance Oracles with Improved Bounds. In *Proceedings of the 47th Symposium on Theory of Computing (STOC)*, pages 1–10, 2015. doi:10.1145/2746539.2746562.

[CLPR12] Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. f-Sensitivity Distance Oracles and Routing Schemes. *Algorithmica*, 63:861–882, 2012. doi:10.1007/s00453-011-9543-0.

[CZ22] Shiri Chechik and Tianyi Zhang. Nearly 2-Approximate Distance Oracles in Subquadratic Time. In *Proceedings of the 33rd Symposium on Discrete Algorithms (SODA)*, pages 551–580, 2022. doi:10.1137/1.9781611977073.26.

[DG24] Dipan Dey and Manoj Gupta. Nearly Optimal Fault Tolerant Distance Oracle. In *Proceedings of the 56th Symposium on Theory of Computing (STOC)*, 2024. To appear.

[DP09] Ran Duan and Seth Pettie. Dual-Failure Distance and Connectivity Oracles. In *Proceedings of the 20th Symposium on Discrete Algorithms (SODA)*, pages 506–515, 2009. URL: https://dl.acm.org/doi/10.5555/545381.545490.

[DR22] Ran Duan and Hanlin Ren. Maintaining Exact Distances under Multiple Edge Failures. In *Proceedings of the 54th Symposium on Theory of Computing (STOC)*, pages 1093–1101, 2022. doi:10.1145/3519935.3520002.

[DT02] Camil Demetrescu and Mikkel Thorup. Oracles for Distances Avoiding a Link-Failure. In *Proceedings of the 13th Symposium on Discrete Algorithms (SODA)*, pages 838–843, 2002. URL: https://dl.acm.org/doi/10.5555/545381.545490.

[DTCR08] Camil Demetrescu, Mikkel Thorup, Rezaul A. Chowdhury, and Vijaya Ramachandran. Oracles for Distances Avoiding a Failed Node or Link. *SIAM Journal on Computing*, 37:1299–1318, 2008. doi:10.1137/S0097539705429847.

[Erd64] Paul Erdős. Extremal Problems in Graph Theory. *Theory of Graphs and its Applications*, pages 29–36, 1964.

[GR21] Yong Gu and Hanlin Ren. 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)*, pages 76:1–76:20, 2021. doi:10.4230/LIPIcs.ICALP.2021.76.

[GV20] Fabrizio Grandoni and Virginia Vassilevska Williams. Faster Replacement Paths and Distance Sensitivity Oracles. *ACM Transaction on Algorithms*, 16:15:1–15:25, 2020. doi:10.1145/3365835.

[Kin99] Valerie King. Fully Dynamic Algorithms for Maintaining All-Pairs Shortest Paths and Transitive Closure in Digraphs. In *Proceedings of the 40th Symposium on Foundations of Computer Science (FOCS)*, pages 81–91, 1999. doi:10.1109/SFFCS.1999.814580.

[Knu17] Mathias Bæk Tejs Knudsen. Additive Spanners and Distance Oracles in Quadratic Time. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 64:1–64:12, 2017. doi:10.4230/LIPIcs.ICALP.2017.64.

[KP21] Karthik C.S. and Merav Parter. Deterministic Replacement Path Covering. In *Proceedings of the 32nd Symposium on Discrete Algorithms (SODA)*, pages 704–723, 2021. doi:10.1137/1.9781611976465.44.

[KS23] Adam Karczmarz and Piotr Sankowski. Sensitivity and dynamic distance oracles via generic matrices and frobenius form. In *Proceedings of the 64th Symposium on Foundations of Computer Science (FOCS)*, pages 1745–1756, 2023. doi:10.1109/FOCS57990.2023.00106.

[PR11] Ely Porat and Liam Roditty. Preprocess, Set, Query! In *Proceedings of the 19th European Symposium on Algortihms (ESA)*, pages 603–614, 2011. doi:10.1007/978-3-642-23719-5_51.

[PR14] Mihai Pătraşcu and Liam Roditty. Distance Oracles Beyond the Thorup-Zwick Bound. *SIAM Journal on Computing*, 43:300–311, 2014. doi:10.1137/11084128X.

[PRT12] Mihai Pătraşcu, Liam Roditty, and Mikkel Thorup. A New Infinity of Distance Oracles for Sparse Graphs. In *Proceedings of the 53rd Symposium on Foundations of Computer Science (FOCS)*, pages 738–747, 2012. doi:10.1109/FOCS.2012.44.

[Ren22] Hanlin Ren. Improved Distance Sensitivity Oracles with Subcubic Preprocessing Time. *Journal of Computer and System Sciences*, 123:159–170, 2022. doi:10.1016/j.jcss.2021.08.005.

[RZ12] Liam Roditty and Uri Zwick. Replacement Paths and k Simple Shortest Paths in Unweighted Directed Graphs. *ACM Transaction on Algorithms*, 8:33:1–33:11, 2012. doi:10.1145/2344422.2344423.

[Som16] Christian Sommer. All-pairs Approximate Shortest Paths and Distance Oracle Preprocessing. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, 2016. doi:10.4230/LIPIcs.ICALP.2016.55.

[TZ05] Mikkel Thorup and Uri Zwick. Approximate Distance Oracles. *Journal of the ACM*, 52:1–24, 2005. doi:10.1145/1044731.1044732.

[WN13] Christian Wulff-Nilsen. Approximate Distance Oracles with Improved Query Time. In *Proceedings of the 24th Symposium on Discrete Algorithms (SODA)*, pages 539–549, 2013. doi:10.1137/1.9781611973105.39.

[WY13] Oren Weimann and Raphael Yuster. Replacement Paths and Distance Sensitivity Oracles via Fast Matrix Multiplication. *ACM Transactions on Algorithms*, 9:14:1–14:13, 2013. doi:10.1145/2438645.2438646.