Fault-Tolerant Approximate Distance Oracles with a Source Set

Dipan Dey
Tata Institute of Fundamental Research, Mumbai, India

Telikepalli Kavitha
Tata Institute of Fundamental Research, Mumbai, India

Abstract

Our input is an undirected weighted graph $G = (V, E)$ on n vertices along with a source set $S \subseteq V$. The problem is to preprocess G and build a compact data structure such that upon query $Qu(s, v, f)$ where $(s, v) \in S \times V$ and f is any faulty edge, we can quickly find a good estimate (i.e., within a small multiplicative stretch) of the s-v distance in $G - f$. We use a fault-tolerant ST-distance oracle from the work of Bilò et al. (STACS 2018) to construct an $S \times V$ approximate distance oracle or *sourcewise* approximate distance oracle of size $\widetilde{O}(|S|n + n^{3/2})$ with multiplicative stretch at most 5. We construct another fault-tolerant sourcewise approximate distance oracle of size $\widetilde{O}(|S|n + n^{4/3})$ with multiplicative stretch at most 13. Both the oracles have $O(1)$ query answering time.

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

Keywords and phrases Weighted graphs, approximate distances, fault-tolerant data structures

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.27

Funding Supported by the Department of Atomic Energy, Government of India, under project no. RTI4001.

1 Introduction

The problem of computing distances between all pairs of vertices in a given graph $G = (V, E)$ with positive edge weights is a fundamental problem in graph algorithms. The problem here is to preprocess G and build a compact data structure (called a *distance oracle*) that can quickly answer distance queries for any pair of vertices. As is often the case with real-world networks like routing networks or road networks, links may fail or roads may be temporarily blocked. Thus we have to allow for the case of faulty edges. Since several links are unlikely to fail simultaneously, we consider the case of a single edge failure.

Instead of recomputing distances from scratch for all pairs of vertices after an edge has failed, the problem is to build a resilient data structure that can answer distance queries between vertices after a single edge failure. Furthermore, we assume there is a specific set $S \subseteq V$ of sources, e.g., S is a set of starting locations on a road network or S is a set of source nodes in a routing network. Thus we are interested in distances only for pairs $(s, v) \in S \times V$. Suppose $|S| \ll n$, say $O(n^\epsilon)$ for some $\epsilon \in (0, 1)$. Then it feels wasteful to build a fault-tolerant distance oracle that maintains distances for all pairs of vertices. Another application is Vickrey pricing [21], with the objective of determining, for every $(s, v) \in S \times V$ where S is a given subset of V and every edge f, how much the distance from s to v increases if f were to fail.

Thus the distance oracle has to process queries of the form (s, v, f) where s is the source, v is the destination, and f is the failed edge. Upon query $Qu(s, v, f)$, the oracle has to return the s to v distance in $G - f$, where $G - f$ is the graph obtained by removing edge f from G. So our problem is the following (as described below).

45th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2025).
Editors: C. Aiswarya, Ruta Mehta, and Subhajit Roy; Article No. 27; pp. 27:1–27:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- Preprocess G and build a compact data structure that can quickly answer distance queries for any pair in $S \times V$ when an edge fails. Hence our distance oracle has to answer queries $Qu(s, v, f)$ where $s \in S, v \in V$, and f is the failed edge.

This data structure is called a *single edge fault-tolerant sourcewise distance oracle*. As discussed below, for undirected unweighted graphs, a single edge fault-tolerant sourcewise exact distance oracle of size $\tilde{O}(n^{3/2}\sqrt{|S|})$ with $\tilde{O}(1)$ query time is known [20]. Our goal is to design more compact distance oracles (for sublinear sets S) in weighted graphs. Furthermore, for the sake of space efficiency, we are ready to relax *exactness*. Thus the problem we consider is to design a compact fault-tolerant sourcewise *approximate* distance oracle.

Recall that "sourcewise" captures the fact that we are interested in distances between pairs $(s, v) \in S \times V$. For any pair $(s, v) \in S \times V$ and $f \in E$, let $||sv \diamond f||$ denote the distance from s to v in $G - f$. A fault-tolerant approximate distance oracle is said to have *multiplicative stretch* α if the distance $d_{G-f}(s, v)$ returned by the oracle on query $Qu(s, v, f)$ is sandwiched between the actual distance and α times the actual distance, i.e., $||sv \diamond f|| \leq d_{G-f}(s, v) \leq \alpha \cdot ||sv \diamond f||$. We show the following result.

▶ **Theorem 1.** *Let $G = (V, E)$ be an undirected graph on n vertices with positive edge weights. For any $S \subseteq V$, a fault-tolerant sourcewise approximate distance oracle with multiplicative stretch at most 5 and size $\widetilde{O}(|S|n + n^{3/2})$ can be constructed in polynomial time such that $Qu(s, v, f)$ where $(s, v) \in S \times V$ and $f \in E$ can be answered in constant time.*

Note that our oracle has size $\widetilde{O}(n^{3/2})$ when $|S| = O(\sqrt{n})$. For smaller sets S, we show a sparser fault-tolerant sourcewise approximate distance oracle at the expense of a larger stretch. Its query answering time is also $O(1)$.

▶ **Theorem 2.** *Let $G = (V, E)$ be an undirected graph on n vertices with positive edge weights. For any $S \subseteq V$, a fault-tolerant sourcewise approximate distance oracle with multiplicative stretch at most 13 and size $\widetilde{O}(|S|n + n^{4/3})$ can be constructed in polynomial time such that $Qu(s, v, f)$ where $(s, v) \in S \times V$ and $f \in E$ can be answered in constant time.*

Thus the above oracle has size $\widetilde{O}(n^{4/3})$ when $|S| = O(n^{1/3})$. The work of Bilò, Gualà, Leucci, and Proietti on multiple-edge fault-tolerant approximate shortest path trees [11] in undirected weighted graphs with a *single source* (so $|S| = 1$) implies a multiple-edge fault-tolerant *sourcewise*[1] (so S is any subset of V) approximate distance oracle of size $\widetilde{O}(|S|n)$ with a stretch of 3. Thus their oracle is sparser than our oracles when $|S|$ is small and it also achieves a better stretch. However, our query answering time is $O(1)$ while theirs is $O(\log^2 n)$, where n is the number of vertices. Our algorithms are truly simple while their techniques are quite involved.

As mentioned above, the problem of constructing fault-tolerant sourcewise exact distance oracles in undirected unweighted graphs has been studied earlier. Also, in undirected weighted graphs, the problem of constructing fault-tolerant *single source* (so $|S| = 1$) exact distance oracles has been studied. We discuss these results below.

Background. The first fault-tolerant exact distance oracle was designed by Demetrescu and Thorup in 2002 [14] and it was for directed weighted graphs. Their oracle handles single edge failures and has size $O(n^2 \log n)$ with $O(1)$ query time. After this result, there has

[1] Unfortunately, we were unaware of this work at the time of paper submission. We thank Manoj Gupta for bringing this paper to our attention.

been a long line of research on the problem of efficiently constructing single edge/vertex fault-tolerant exact distance oracles. Ignoring preprocessing time, the most space-efficient oracle is by Duan and Zhang [19] with size $O(n^2)$ and query time $O(1)$. Thus it shaves off the $\log n$ factor from the size of the original oracle.

Fault-tolerant sourcewise distance oracles. For undirected unweighted graphs, Gupta and Singh [20] designed a single edge fault-tolerant sourcewise exact distance oracle of size $\tilde{O}(n^{3/2}\sqrt{|S|})$ with $\tilde{O}(1)$ query time and source set S. In undirected graphs with edge weights in the range $\{1, 2, \ldots, M\}$, Bilò, Cohen, Friedrich and Schirneck [10] designed a fault-tolerant single source exact distance oracle. This oracle handles single edge failures and has size $\tilde{O}(n^{3/2}\sqrt{M})$ with query time $\tilde{O}(1)$. For undirected unweighted graphs, Dey and Gupta [15] designed a different oracle with the same space and query time bounds as in [10], but with a faster preprocessing time.

ST-distance oracles in directed graphs. In directed weighted graphs, Bilò, Choudhary, Gualà, Leucci, Parter and Proietti [9] designed a fault-tolerant ST-distance oracle, i.e., it maintains exact distances for all pairs in $S \times T$, for given vertex subsets S and T. It handles single edge failures and has size $\widetilde{O}((|S| + |T|)n)$ with $O(1)$ query time, where n is the number of vertices. They also designed a fault-tolerant ST-distance oracle in unweighted directed graphs of size $\widetilde{O}(n\sqrt{|S||T|})$ with query time $O(\sqrt{|S||T|})$. Furthermore, they showed a fault-tolerant *ST-approximate* distance oracle in directed unweighted graphs that returns in constant time a distance estimate stretched by an additive term. In particular, when $|S| = O(\sqrt{n})$, their oracle has size $\widetilde{O}(n^{3/2})$ and additive stretch $\widetilde{O}(\sqrt{n})$.

Fault-tolerant approximate distance oracles. Approximate distance oracles that provide distances within a small multiplicative stretch for all vertex pairs have been extensively studied. Table 1 summarizes results for fault-tolerant approximate distance oracles in directed/undirected graphs. Note that the stretch here is multiplicative, except for the last row where the stretch has an additive term as well.

Table 1 A table listing the works related to fault-tolerant approximate distance oracles where D is the diameter of the graph.

Graph	Faults	Stretch	Size	Query time	Ref
Undirected Weighted	$c \geq 1$	$(8k-2)(c+1)$, $k \geq 1$ integer	$O(ckn^{1+1/k}\log(nM))$, M is the max edge wt	$\tilde{O}(c)$	[13]
Undirected Unweighted	$c = 1$	$(2k-1)(1+\epsilon)$, $k \geq 1$ integer and $\epsilon > 0$	$\tilde{O}\left(\frac{k^5}{\epsilon^4}n^{1+1/k}\right)$	$O(1)$	[2]
Undirected Weighted	$c = o\left(\frac{\log n}{\log\log n}\right)$	$(1+\epsilon)$	$O(n^2(\log D/\epsilon)^c c\log D)$	$O(c^5\log D)$	[12]
Directed Unweighted	$c \geq 2$	$(3+\epsilon)$	$\tilde{O}(n^{2-\frac{\alpha}{c+1}}/\epsilon)(\log n/\epsilon)^c)$ where $\alpha \in (0, 1/2)$ and $\epsilon > 0$	$O(n^\alpha/\epsilon^2)$	[6]
Undirected Weighted	$c = o\left(\frac{\log n}{\log\log n}\right)$	$(2k-1)$ where $k \geq 1$ integer	$O(n^{1+\frac{1}{k}+\alpha+o(1)})$ where $\alpha \in (0,1)$	$O(n^{1+\frac{1}{k}-\frac{\alpha}{k(c-1)}})$	[8]
Undirected Unweighted	$c = o\left(\frac{\log n}{\log\log n}\right)$	$(\frac{k+1}{k})(1+\epsilon)$ with additive stretch of 2, $k \geq 1$ integer and $\epsilon \geq 0$	$O\left(\frac{n^{2-\frac{\gamma}{(k+1)(c+1)}+o(1)}}{\epsilon^{c+2}}\right)$ where $\gamma \in \left(0, \frac{k+1}{2}\right)$	$O(n^\gamma/\epsilon^2)$	[7]

For single edge faults, note that Chechik, Langberg, Peleg, and Roditty [13] showed an approximate distance oracle with stretch 12 and size $\widetilde{O}(n^{3/2})$ and another with stretch 28 and size $\widetilde{O}(n^{4/3})$. In comparison to this, Theorem 1 shows a sourcewise approximate distance oracle with stretch 5 and size $\widetilde{O}(|S|n+n^{3/2})$ and Theorem 2 shows a sourcewise approximate distance oracle with stretch 13 and size $\widetilde{O}(|S|n+n^{4/3})$. Thus for small sets S, our oracles are as sparse and have smaller stretch. Note that for single faults and every $k \geq 1$, approximate distance oracles by Baswana and Khanna [2] are almost as sparse as the oracles in [13] and have significantly smaller stretch. However these oracles work only for unweighted graphs.

It is an open problem if our construction can be generalized to work for all integers k, in other words, to show a sourcewise approximate distance oracle of size $\widetilde{O}(|S|n + n^{1+1/k})$ and stretch $8k - 3$ with $O(1)$ query answering time for $k \geq 3$. Our results show such a construction for $k = 1, 2$. Note that the remaining approximate distance oracles in Table 1 have superconstant query time, so our oracles cannot directly be compared with them.

Our techniques. Our algorithms are simple to describe and use the ST-distance oracle by Bilò, Choudhary, Gualà, Leucci, Parter and Proietti [9]. Their oracle uses *landmark* vertices, i.e., vertices picked uniformly at random from the vertex set V (originally used by Bernstein and Karger [4]).[2] The oracle in Theorem 1 uses this ST-distance oracle for the given source set S and $T = S \cup \mathcal{L}$, where $\mathcal{L}$ is our landmark vertex set. The oracle in Theorem 2 is based on the same idea, however there are two levels of sampling here: so we have two landmark vertex sets $\mathcal{L}_2 \subseteq \mathcal{L}_1$. Theorem 1 and Theorem 2 are proved in Section 3 and Section 4, respectively. We discuss preliminaries in Section 2 and conclude in Section 5.

2 Preliminaries

This section describes the notation that will be used in the rest of the paper and also gives a sketch of the ST-distance oracle from [9]. Our input is an undirected graph $G = (V, E)$ with positive edge weights as given by $\mathsf{wt} : E \to \mathbb{R}_+$. For any path ρ in G:

- let $||\rho||$ be the *length* of ρ, i.e., $||\rho|| = \sum_{e\in\rho} \mathsf{wt}(e)$;
- let $|\rho|$ be the *hop length* of ρ, i.e., the number of edges in ρ.

For any $(u, v) \in V \times V$, a shortest path between u and v is a path of minimum length between u and v. We assume the shortest path between any two vertices in the graph is unique. This property can be achieved by random perturbation of the given edge weights (e.g., see [22]). The property of unique shortest paths was also used in [5, 17, 18, 20, 21]. We denote the shortest path from u to v by uv. Thus $||uv||$ is the distance between u and v in G and $|uv|$ is the hop length between u and v in G.

- Let $G - f = (V, E \setminus \{f\})$ be the graph obtained after deleting edge f from the graph G. As in G, we assume there is a unique shortest path between any pair of vertices in $G - f$.
- For any $(u, v) \in V \times V$ and $f \in E$, let $uv \diamond f$ be the shortest path between u and v in $G - f$. So $||uv \diamond f||$ is the distance between u and v in $G - f$.

The concept of *landmark* vertices will be key to our distance oracles.

▶ **Definition 3** (Landmark Vertex Set, $\mathcal{L}$). *Sample each vertex in G independently with probability p. The selected set (call it $\mathcal{L}$) of vertices is the landmark vertex set.*

[2] To the best of our knowledge, the name "landmark" vertices was first used by Dey and Gupta [16].

The probability p in Definition 3 will be set to different values in Section 3 and Section 4. The following proposition on the landmark vertex set $\mathcal{L}$ will be very useful to us.

▶ **Proposition 4.** *With high probability, for any pair of vertices u and v, if $|uv| \geq \lfloor \frac{3\ln n}{p} \rfloor$ then there is at least one landmark vertex on uv.*

Proof. Since each vertex in G is sampled independently with probability p, for any pair of vertices u and v, the probability that there is *no* landmark vertex on uv is $(1-p)^k$ where $k = |uv| + 1$ is the number of vertices on uv. Because $|uv| \geq \lfloor \frac{3\ln n}{p} \rfloor$, the probability that there is no landmark vertex on uv is at most:

$$(1-p)^{\frac{3\ln n}{p}} \quad \leq \quad \left(\frac{1}{e}\right)^{3\ln n} \quad \leq \quad \frac{1}{n^3}.$$

Thus for any pair of vertices u and v with $|uv| \geq \lfloor \frac{3\ln n}{p} \rfloor$, the probability that there is no landmark vertex on uv is at most $1/n^3$. Hence the probability that there is some pair $(x,y) \in V \times V$ with $|xy| \geq \lfloor \frac{3\ln n}{p} \rfloor$ such that there is no landmark vertex on xy is at most $\binom{n}{2}/n^3 \leq 1/n$. Thus with probability at least $1 - 1/n$, it is the case that for every pair $(u,v) \in V \times V$ with $|uv| \geq \lfloor \frac{3\ln n}{p} \rfloor$, there is at least one landmark vertex on uv. ◀

Since each vertex in G is sampled with probability p, the expected size of $\mathcal{L}$ is np. We will set $p = n^{-\delta}$ for some $\delta \in (0,1)$ in Section 3 and Section 4. Thus with high probability, we will have $|\mathcal{L}| \leq 2np$ (by Chernoff bound).

- If either $|\mathcal{L}| > 2np$ or there exists a pair of vertices u, v with $|uv| \geq \lfloor \frac{3\ln n}{p} \rfloor$ such that there is no vertex of $\mathcal{L}$ on uv then we will repeat the step of sampling vertices and construct another landmark vertex set such that both these properties hold for the set obtained.

Thus we will assume that $|\mathcal{L}| = O(np)$ and every pair of vertices u, v with $|uv| \geq \lfloor \frac{3\ln n}{p} \rfloor$ has at least one vertex of $\mathcal{L}$ on uv. The expected number of trials to obtain a desired landmark set $\mathcal{L}$ is $O(1)$.

Fault-Tolerant ST-Distance Oracle. We now briefly discuss the algorithm of Bilò et al.[9] to construct a fault-tolerant exact distance oracle in G for pairs $(s,t) \in S \times T$, where $S \subseteq V$ and $T \subseteq V$ are part of the input. Fix a pair $(s,t) \in S \times T$ and let $f = (a,b)$ be any edge on st. Let ℓ and ℓ' be the two landmark vertices on st closest to a and b on as and bt, respectively.

There are 3 cases with respect to the replacement path $st \diamond f$: (i) $st \diamond f$ goes through ℓ, (ii) $st \diamond f$ goes through ℓ', (iii) $st \diamond f$ goes through neither ℓ nor ℓ'. Their algorithm builds tables to deal with each of these cases. Figure 1 captures the main idea.

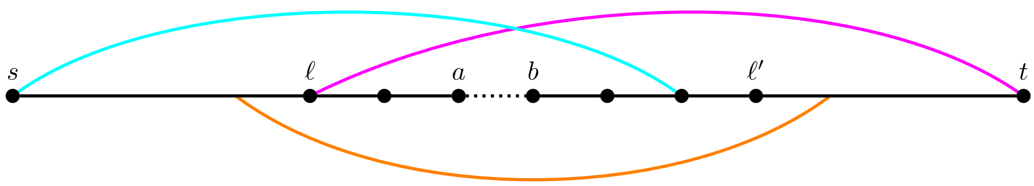

Figure 1 The replacement path $st \diamond f$ where $f = (a,b)$ in case (i) is $s\ell$ followed by the magenta path $\ell t \diamond f$; in case (ii) it is the blue path $s\ell' \diamond f$ followed by $\ell' t$ and in case (iii) $st \diamond f$ avoids both ℓ and ℓ' - so the orange path is part of $st \diamond f$.

The following theorem from [9] will be used in our algorithms.

▶ **Theorem 5** ([9]). *An n-vertex directed or undirected weighted graph G for given subsets S and T of V can be preprocessed in polynomial time to compute a data structure of size $O((|S| + |T|)n \log n)$ that given any pair $(s,t) \in S \times T$ and any failing edge f can report $||st \diamond f||$ in constant time.*

3 Sourcewise Approximate Distance Oracle for a Single Edge Fault

Our input is an undirected weighted graph $G = (V, E)$ with a positive weight function $\mathsf{wt} : E \to \mathbb{R}_+$ and a subset $S \subseteq V$ of sources. The goal is to build a compact data structure that can answer distance queries $Qu(s, v, f)$ within a small multiplicative stretch, where $s \in S, v \in V$ and f is the edge fault.

Landmark vertex set $\mathcal{L}$. Recall Definition 3 on landmark vertices. Let us sample each vertex independently with probability $p = (3 \ln n)/\sqrt{n}$ to obtain our landmark vertex set $\mathcal{L}$. The following two properties hold (see Proposition 4); otherwise we resample to obtain another landmark vertex set $\mathcal{L}$ so that the following two properties hold.

- $|\mathcal{L}| = O(\sqrt{n} \log n)$.
- For any pair of vertices u and v: if $|uv| \geq \lfloor \sqrt{n} \rfloor$, then there is at least one landmark vertex on uv.

Our algorithm. On input $G = (V, E)$ and $S \subseteq V$, the first step of our algorithm is to build the above landmark vertex set $\mathcal{L}$. We then compute shortest path trees $\mathcal{T}(u)$ rooted at u for all $u \in S \cup \mathcal{L}$. Along with every vertex v, the tree $\mathcal{T}(u)$ also has the two attributes $||uv||$ and $|uv|$, i.e., the length and the hop length of uv.

Our algorithm constructs the ST-exact distance oracle from [9] fixing the source set S and destination set $T = S \cup \mathcal{L}$. For each $v \in V$, let t_v be the vertex in T that is closest to v, where ties are broken arbitrarily. We maintain the lengths of replacement paths $vt_v \diamond f$ for each edge $f \in vt_v$. Our algorithm is described below.

1. Obtain the landmark vertex set $\mathcal{L}$.
2. For each $u \in S \cup \mathcal{L}$ do: compute the shortest path tree $\mathcal{T}(u)$ rooted at u in $G = (V, E)$.
3. Use Theorem 5 to construct an ST-exact distance oracle for the given source set S and target set $T = \mathcal{L} \cup S$ in $G = (V, E)$.
4. For every $v \in V$ in the graph $G = (V, E)$ do:
   - Identify the nearest vertex to v in the target set $T = \mathcal{L} \cup S$. Call this vertex t_v.
   - For $1 \leq i \leq |vt_v|$ do:
     - Let f_i be the i-th edge from t_v on vt_v.
     - Compute the distance $||vt_v \diamond f_i||$ between v and t_v in $G - f_i$.
     - Set $\mathsf{Dist}_T[v, i] = ||vt_v \diamond f_i||$.

Query answering algorithm. In response to the query $Qu(s, v, f)$, the query answering algorithm first checks if $f \in sv$. This check can be done efficiently via LCA queries. Given a rooted tree $\mathcal{T}$ and a pair of vertices x, y in the tree $\mathcal{T}$, recall that $\mathsf{LCA}_{\mathcal{T}}(x, y)$ is the least common ancestor of x and y in tree $\mathcal{T}$.

Observe that $f = (a, b) \in sv$ if and only if the answer to the following three questions is "yes" where $\mathcal{T}(s)$ is the shortest path tree in G rooted at s.

- Is $\mathsf{LCA}_{\mathcal{T}(s)}(v, a)$ equal to a?
- Is $\mathsf{LCA}_{\mathcal{T}(s)}(v, b)$ equal to b?
- Is $|sa| + 1 = |sb|$ or is $|sb| + 1 = |sa|$?

A "yes" answer to the first two questions implies that both a and b are vertices on the path sv. Moreover, a and b are adjacent to each other on sv if and only if the answer to the third question is "yes". Recall that for any vertex w, $|sw|$ is the hop length between s and w, i.e., the number of edges in sw.

Given a tree $\mathcal{T}$, there is a linear time algorithm to build an $O(n)$ size data structure such that LCA queries on $\mathcal{T}$ can be answered in $O(1)$ time [3]. Recall that for every vertex w, the hop length $|sw|$ is stored along with w in $\mathcal{T}(s)$. Thus $|sa|$ and $|sb|$ can be retrieved in $O(1)$ time. Hence the query answering algorithm can determine in $O(1)$ time if $f \in sv$ or not. The query answering algorithm will return $||sv||$ if $f \notin sv$ (see Figure 2). Recall that the distance $||sv||$ is also stored along with v in $\mathcal{T}(s)$.

Figure 2 Here $f = (a, b) \notin sv$, so the path sv is undisturbed by the edge fault f.

If $f \in sv$, then the query answering algorithm looks up the identity of t_v, which is the nearest vertex in T to v. Via LCA queries on $\mathcal{T}(t_v)$, we can determine if $f = (a, b) \in vt_v$ or not. If not, then $||vt_v \diamond f|| = ||vt_v||$. So let us assume $f \in vt_v$.

Assume without loss of generality that b is closer than a to t_v, i.e., $|at_v| = |bt_v| + 1$ (see Figure 2). Let i be the index such that a is the i-th vertex from t_v on vt_v. Then $||vt_v \diamond (a, b)|| = \mathsf{Dist}_T[v, i]$. Recall that the attribute $i = |at_v|$ is stored along with a in $\mathcal{T}(t_v)$.

- The query answering algorithm returns $||vt_v \diamond f|| + ||st_v \diamond f||$, where $||vt_v \diamond f|| = \mathsf{Dist}_T[v, i]$.

Note that the distance $||st_v \diamond f||$ is obtained by querying the ST-distance oracle. Thus the query answering time is $O(1)$. We show below that $||vt_v \diamond f|| + ||st_v \diamond f|| \le 5||sv \diamond f||$.

▶ **Lemma 6.** *For any $(s, v) \in S \times V$ and $f \in E$, our algorithm returns an estimate for the s-v distance in $G - f$ with a multiplicative stretch of at most 5 in constant time.*

Proof. It follows from the discussion above that the query answering time is $O(1)$. Since the query answering algorithm will return $||sv||$ if $f \notin sv$ (see Figure 2), let us assume $f \in sv$. Then the distance estimate returned by the query answering algorithm in response to query $Qu(s, v, f)$ is $||st_v \diamond f|| + ||vt_v \diamond f||$ where t_v is the nearest vertex in T to v. We now bound the sum $||st_v \diamond f|| + ||vt_v \diamond f||$. Consider the following four cases.

1. $f \in st_v$ and $f \in vt_v$. This means the edge f belongs to the shortest path between t_v and the least common ancestor of s and v in $\mathcal{T}(t_v)$ (see Figure 2). However then $f \notin sv$, contradicting our assumption that $f \in sv$.
2. $f \in st_v$ and $f \notin vt_v$. The query answering algorithm will determine via LCA queries that $f \notin vt_v$, so $||vt_v \diamond f|| = ||vt_v||$. Let us bound $||st_v \diamond f||$. The graph $G - f$ has an s-t_v path obtained by stitching the paths $sv \diamond f$ and vt_v, i.e., the path $sv \diamond f$ followed by vt_v. So $||st_v \diamond f|| \le ||sv \diamond f|| + ||vt_v||$. Hence the distance returned is at most $||sv \diamond f|| + 2||vt_v||$.
   - Because t_v is the closest vertex in $T = \mathcal{L} \cup S$ to v, we have $||vt_v|| \le ||vs||$. Hence the distance returned is at most $||sv \diamond f|| + 2||sv|| \le 3||sv \diamond f||$. So the stretch is at most 3 in this case.

3. $f \notin st_v$ and $f \in vt_v$. Since $f \notin st_v$, we have $||st_v \diamond f|| = ||st_v||$. Let us bound $||vt_v \diamond f||$. Since the graph $G - f$ has a vt_v path obtained by stitching $sv \diamond f$ and st_v, we have $||vt_v \diamond f|| \leq ||sv \diamond f|| + ||st_v||$ (see Figure 3). Thus the distance returned by the oracle is at most $||sv \diamond f|| + 2||st_v||$.
   - Observe that $||st_v|| \leq ||sv|| + ||vt_v|| \leq 2||sv||$. Hence the distance returned is at most $||sv \diamond f|| + 4||sv|| \leq 5||sv \diamond f||$. Thus the stretch is at most 5 in this case.

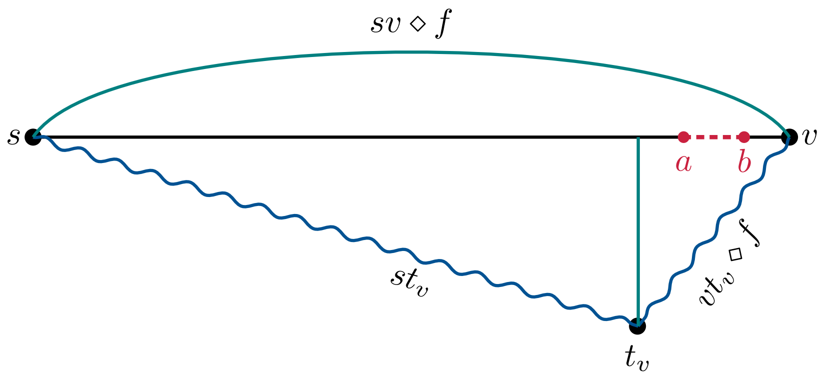

Figure 3 The oracle returns a distance estimate $\leq 2||st_v|| + ||sv \diamond f||$, so the stretch is ≤ 5.

4. $f \notin st_v$ and $f \notin vt_v$. The query answering algorithm will return $||st_v \diamond f|| + ||vt_v \diamond f|| = ||st_v|| + ||vt_v||$ in this case. We have $||vt_v|| \leq ||vs||$ and we also have $||st_v|| \leq ||sv|| + ||vt_v|| \leq 2||sv||$. Thus the stretch is at most 3 in this case.

This finishes the proof of the lemma. ◀

Data structures constructed. Our algorithm constructs in step 3 all the data structures constructed by the ST-distance oracle algorithm. Thus we have access to $||st \diamond f||$ for every $(s, t) \in S \times T$ and $f \in E$. Our oracle also has the table Dist_T that stores $||vt_v \diamond f||$ between v and t_v in $G - f$, for each $v \in V$ and edge $f \in vt_v$. Let us bound the size of our oracle.

▶ **Lemma 7.** *The size of the data structures constructed by our algorithm is* $\widetilde{O}(|S|n + n^{3/2})$.

Proof. The size of T is $\widetilde{O}(|S| + \sqrt{n})$. So the sizes of all the shortest path trees constructed in step 2 is $\widetilde{O}(|S|n + n^{3/2})$. Similarly, the size of the ST-oracle constructed in step 2 is $\widetilde{O}(|S|n + n^{3/2})$ (by Theorem 5). Furthermore, the data structure used to answer LCA queries on each shortest path tree $\mathcal{T}(u)$ has size $O(n)$. Since $u \in S \cup \mathcal{L}$, these data structures also take up space $\widetilde{O}(|S|n + n^{3/2})$.

It follows from the property of our landmark set $\mathcal{L}$ that for any vertex v, we have $\min\{|v\ell| : \ell \in \mathcal{L}\} \leq \lfloor\sqrt{n}\rfloor$. So for each vertex v, we have $||vt_v \diamond f||$ stored for at most $\lfloor\sqrt{n}\rfloor$ many edges f, where t_v is the nearest vertex in T to v. Thus the size of Dist_T is $O(n \cdot \sqrt{n}) = O(n^{3/2})$. This finishes the proof of the lemma. ◀

It is easy to see that our algorithm runs in expected polynomial time. Recall that we used randomization to construct the set $\mathcal{L}$. By blowing up the size of $\mathcal{L}$ by a factor of $\log n$, the construction of $\mathcal{L}$ can be made deterministic (see [9, Lemma 1]). Thus Theorem 1 follows. We restate it below for convenience.

▶ **Theorem 1.** *Let $G = (V, E)$ be an undirected graph on n vertices with positive edge weights. For any $S \subseteq V$, a fault-tolerant sourcewise approximate distance oracle with multiplicative stretch at most 5 and size $\widetilde{O}(|S|n + n^{3/2})$ can be constructed in polynomial time such that $Qu(s, v, f)$ where $(s, v) \in S \times V$ and $f \in E$ can be answered in constant time.*

▶ Remark 8. The query answering algorithm can return not only an approximate estimate of the distance $||sv \diamond f||$, but also the corresponding approximate shortest path in a succinct form. It is known that any replacement path $\rho \diamond f$ is *2-decomposable*, i.e., it is a concatenation of at most 2 shortest paths interleaved with at most 1 edge [1].

So along with any distance $||st_v \diamond f||$ (similarly, $||vt_v \diamond f||$), we could also store the corresponding replacement paths in 2-decomposable form. Thus the query answering algorithm can return the corresponding s-v approximate shortest path as the union of two replacement paths $\rho_1 = st_v \diamond f$ and $\rho_2 = vt_v \diamond f$, each in 2-decomposable form, say, $\rho_1 = \langle s, x, y, t_v \rangle$ and $\rho_2 = \langle v, x', y', t_v \rangle$. This will mean ρ_1 is the shortest path in G between s and x followed by the edge (x, y), and the shortest path in G between y and t_v, similarly for ρ_2.

4 A Sparser Fault-Tolerant Sourcewise Approximate Distance Oracle

In this section, we present another fault-tolerant sourcewise approximate distance oracle for single edge faults. As before, the input is an undirected weighted graph $G = (V, E)$ with a positive weight function $\mathsf{wt} : E \to \mathbb{R}_+$ and a subset $S \subseteq V$ of sources. Our goal is to build a sparser data structure that can answer distance queries $Qu(s, v, f)$ within a small multiplicative stretch. For sets S of size $o(\sqrt{n})$, the oracle in this section will be sparser than the one in Section 3.

Our algorithm. We will now construct *two* sets $\mathcal{L}_1$ and $\mathcal{L}_2$ of landmark vertices. We will first run the sampling step in Section 2 with $p = (3 \ln n)/n^{1/3}$. Let $\mathcal{L}_1$ be the resulting landmark set. The following properties follow from Section 2 (see Proposition 4).

- $|\mathcal{L}_1| = O(n^{2/3} \log n)$.
- For any pair of vertices u and v: if $|uv| \geq \lfloor n^{1/3} \rfloor$ then there is at least one vertex of $\mathcal{L}_1$ on uv.

After that, we sample each vertex of $\mathcal{L}_1$ with probability $1/n^{1/3}$. Let $\mathcal{L}_2$ be the set of selected vertices. Observe that this 2-step sampling to obtain $\mathcal{L}_2$ is equivalent to running the sampling step in Section 2 with $p = (3 \ln n)/n^{2/3}$ on the entire vertex set V. Thus the following properties follow from Section 2 (see Proposition 4).

- $|\mathcal{L}_2| = O(n^{1/3} \log n)$.
- For any pair of vertices u and v: if $|uv| \geq \lfloor n^{2/3} \rfloor$ then there is at least one vertex of $\mathcal{L}_2$ on uv.

Rather than sampling each vertex of V with probability $p = (3 \ln n)/n^{2/3}$ to get $\mathcal{L}_2$, we did this in two steps so that we have $\mathcal{L}_2 \subseteq \mathcal{L}_1$. Let $T_1 = \mathcal{L}_1 \cup S$ and let $T_2 = \mathcal{L}_2 \cup S$. Our algorithm will use the following notations for any vertex v.

- Let t_v be the vertex in T_1 that is nearest to v.
- Let t'_v be the vertex in T_2 that is nearest to v.

For each $u \in T_2$, we will keep the shortest path tree $\mathcal{T}(u)$ in G rooted at u. However, we cannot afford to keep shortest path trees rooted at each $u \in \mathcal{L}_1$, since that would exceed the desired space bound. Corresponding to each $u \in \mathcal{L}_1$, let $\mathsf{Ball}(u) = \{v \in V : t_v = u\}$ be the set of all vertices v that regard u as their nearest vertex in T_1.

- For each $v \in \mathsf{Ball}(u)$, we will store the path uv.
- Thus we keep a *truncated* shortest path tree $\hat{\mathcal{T}}(u) = \cup_{v \in \mathsf{Ball}(u)} uv$ in G rooted at u for each $u \in \mathcal{L}_1$.

Along with each vertex $v \in \hat{\mathcal{T}}(u)$ where $u \in \mathcal{L}_1$, we also store $|uv|$, i.e., the hop length of uv, and the distance $||uv||$. Similarly, as done in Section 3, along with each vertex $v \in \mathcal{T}(u)$, where $u \in T_2$, we store $|uv|$ and $||uv||$.

Below we describe the steps in our algorithm.

1. Obtain the landmark sets $\mathcal{L}_1$ and $\mathcal{L}_2$, where $\mathcal{L}_2 \subseteq \mathcal{L}_1$, as described above.
2. For each $u \in T_2 = S \cup \mathcal{L}_2$ do: compute the shortest path tree $\mathcal{T}(u)$ rooted at u in G.
3. Use Theorem 5 to construct an ST-exact distance oracle for the given source set S and target set $T = T_2$ in $G = (V, E)$.
4. For each $u \in \mathcal{L}_1$ do: compute the *truncated* shortest path tree $\hat{\mathcal{T}}(u)$ rooted at u in G.
5. For every $v \in V$ do:
   a. Let $t_v \in T_1 = S \cup \mathcal{L}_1$ be the vertex in T_1 that is nearest to v.
   b. For $1 \leq i \leq |vt_v|$ do:
      - Set $\mathsf{Dist}_1[v, i] = ||vt_v \diamond f_i||$ where f_i is the i-th edge from t_v on vt_v.
6. For every $u \in T_1$ do:
   a. Let $t'_u \in T_2$ be the vertex in T_2 that is nearest to u.
   b. For $1 \leq j \leq |ut'_u|$ do:
      - Set $\mathsf{Dist}_2[u, j] = ||ut'_u \diamond f_j||$ where f_j is the j-th edge from t'_u on ut'_u.

Observe that the array $\mathsf{Dist}_1[v, i]$ stores for any vertex v, the distance $||vt_v \diamond f||$ where f is the i-th edge from t_v on the path vt_v. Similarly, the array $\mathsf{Dist}_2[u, j]$ stores for any vertex $u \in T_1$, the distance $||ut'_u \diamond f||$ where f is the j-th edge from t'_u on the path ut'_u.

The query answering algorithm. In response to the query $Qu(s, v, f)$, the query answering algorithm first checks if $f \in sv$. As described in Section 3, this is done by checking the answers to some LCA queries in $\mathcal{T}(s)$. Let us assume $f \in sv$, otherwise the query answering algorithm will return $||sv||$.

Then the query answering algorithm looks up $x = t_v$ and $y = t'_x$. In more detail, (i) x is the closest vertex to v in T_1 and (ii) y is the closest vertex to x in T_2. The query answering algorithm needs to know if $f \in vx$ or not; if so, it also needs to know the index $i \in \{1, \ldots, n^{1/3}\}$ such that f is the i-th edge on xv. As described in Section 3, we can decide if $f \in vx$ or not via LCA queries on the truncated shortest path tree $\hat{\mathcal{T}}(x)$. If so, we can also obtain from $\hat{\mathcal{T}}(x)$ the value i such that f is the i-th edge from x on xv.

Thus the query answering algorithm knows in $O(1)$ time whether $f \in xv$ or not and if so, the index i such that f is the i-th edge from x on xv.

- If $f \notin vx$ then $||vx \diamond f|| = ||vx||$; else $||vx \diamond f|| = \mathsf{Dist}_1[v, i]$.

Recall that we compute $\mathcal{T}(u)$ for all $u \in T_2$. Thus, as described in Section 3, we can efficiently check if $f \in xy$ or not; if so, the algorithm also knows the index j such that f is the j-th edge from y on the path xy.

- If $f \notin xy$ then $||xy \diamond f|| = ||xy||$; else $||xy \diamond f|| = \mathsf{Dist}_2[x, j]$.

Since $s \in S$ and $y \in T_2$ (recall that $T = T_2$), the distance $||ys \diamond f||$ is obtained by querying the ST-distance oracle. Thus the query answering algorithm can obtain $||vx \diamond f||, ||xy \diamond f||$, and $||ys \diamond f||$ in $O(1)$ time. In response to the query $Qu(s, v, f)$, the query answering algorithm returns $||vx \diamond f|| + ||xy \diamond f|| + ||ys \diamond f||$.

We will show in Lemma 9 that our s-v distance estimate in $G - f$ is at most $13||sv \diamond f||$.

▶ **Lemma 9.** *For any $(s, v) \in S \times V$ and $f \in E$, our algorithm returns an s-v distance estimate with stretch ≤ 13 in $G - f$ in $O(1)$ time.*

Proof. Suppose the query is $Qu(s, v, f)$. If $f \notin sv$ then the algorithm returns $||sv||$, thus the stretch is 1 in this case. So assume $f \in sv$. Then the query answering algorithm returns $||vx \diamond f|| + ||xy \diamond f|| + ||ys \diamond f||$, where $x = t_v$ and $y = t'_x$. Let us bound the stretch.

We need to compare the sum $||vx \diamond f|| + ||xy \diamond f|| + ||ys \diamond f||$ with $||sv \diamond f||$. Let us first show the following claim.

▷ Claim 10. We have (i) $||vx|| \le ||vs||$, (ii) $||xy|| \le 2||vs||$, and (iii) $||ys|| \le 4||vs||$.

Proof. It follows from the definition of $T_1 = \mathcal{L}_1 \cup S$ that both x and s are in T_1. Since x is the nearest vertex in T_1 to v, we have $||vx|| \le ||vs||$. Recall that $y = t'_x$. Since y is the closest vertex in T_2 to x, we have $||xy|| \le ||xt'_v||$, i.e., the x-y distance is at most the distance between x and t'_v (recall that t'_v is the nearest vertex in T_2 to v). Furthermore, $||xt'_v|| \le ||xv|| + ||vt'_v||$.

Observe that both $||xv||$ and $||vt'_v||$ are at most $||sv||$ since $s \in T_1 \cap T_2$, so v's distance to its nearest vertex in T_1 and also in T_2 is at most $||sv||$. Thus $||xy|| \le 2||sv||$. So we have $||ys|| \le ||sv|| + ||vx|| + ||xy|| \le ||sv|| + ||sv|| + 2||sv|| = 4||sv||$. ◁

We are now ready to bound $||vx \diamond f|| + ||xy \diamond f|| + ||ys \diamond f||$. There are 8 cases depending on the presence of edge f on various shortest paths.

1. $f \notin vx$ and $f \notin xy$ and $f \notin ys$. Then the algorithm returns $||vx|| + ||xy|| + ||ys||$.
   - It immediately follows from Claim 10 that the s-v distance estimate returned in this case is at most $7||sv|| \le 7||sv \diamond f||$.

2. $f \notin vx$ and $f \notin xy$ and $f \in ys$. Then the algorithm returns $||vx|| + ||xy|| + ||ys \diamond f||$. Since the failed edge f belongs to neither vx nor xy, we have $||sy \diamond f|| \le ||sv \diamond f|| + ||vx|| + ||xy||$. We have $||vx|| + ||xy|| \le 3||sv||$ (by Claim 10).
   - Thus the s-v distance estimate returned in this case is at most $||sv \diamond f|| + 6||sv|| \le 7||sv \diamond f||$ (by Claim 10).

3. $f \notin vx$ and $f \in xy$ and $f \notin ys$. Then the algorithm returns $||vx|| + ||xy \diamond f|| + ||ys||$. Observe that $G - f$ has an x-y path of length at most $||xv|| + ||vs \diamond f|| + ||sy||$. Since $||ys|| \le 4||vs||$ (by Claim 10), this x-y path in $G - f$ is of length at most $||sv|| + ||sv \diamond f|| + 4||sv|| = 5||sv|| + ||sv \diamond f||$.
   - Using Claim 10 to bound $||vx||$ and $||ys||$, the s-v distance estimate returned in this case is at most $||sv|| + 5||sv|| + ||sv \diamond f|| + 4||sv|| = 10||sv|| + ||sv \diamond f|| \le 11||sv \diamond f||$.

4. $f \in vx$ and $f \notin xy$ and $f \notin ys$. Then the algorithm returns $||vx \diamond f|| + ||xy|| + ||ys||$. Observe that $G - f$ has a v-x path of length at most $||vs \diamond f|| + ||sy|| + ||yx||$. This is of length at most $||sv \diamond f|| + 4||sv|| + 2||sv|| = 6||sv|| + ||sv \diamond f||$.
   - Using Claim 10 to bound $||xy||$ and $||ys||$, the s-v distance estimate returned in this case is at most $||sv \diamond f|| + 6||sv|| + 2||sv|| + 4||sv|| = 12||sv|| + ||sv \diamond f|| \le 13||sv \diamond f||$.

5. $f \notin vx$ and $f \in xy$ and $f \in ys$. Consider the shortest path tree $\mathcal{T}(x)$ rooted at x in G. Since $f \notin vx$ and $f \in xy$, the edge $f \in wy$ where $w = \mathsf{LCA}_{\mathcal{T}(x)}(v, y)$. But the edge f also belongs to ys and sv – this is not possible (see Figure 4). Thus this case cannot arise.

6. $f \in vx$ and $f \notin xy$ and $f \in ys$. Consider the shortest path tree $\mathcal{T}(s)$ rooted at s in G and let $z = \mathsf{LCA}_{\mathcal{T}(s)}(v, x)$. Since $f \in sv$ and $f \in vx$, it follows that $f \in zv$. Hence $f \notin sx$. Thus there is a v-x path in $G - f$ of length $||vs \diamond f|| + ||sx||$. Since $||sx|| \le ||sv|| + ||vx|| \le 2||sv||$, this v-x path in $G - f$ has length $||sv \diamond f|| + 2||sv||$.

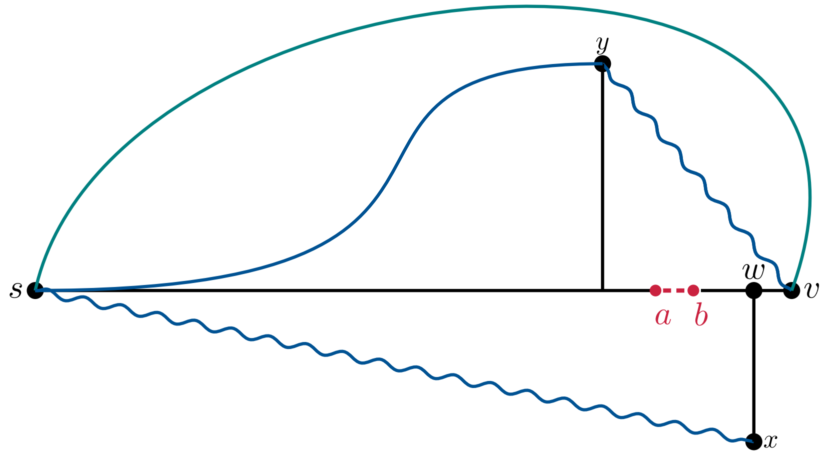

Figure 4 The edge $f = (a, b)$ is supposed to be in the paths sv, xy, and ys, but not in vx. Here $w = \mathsf{LCA}_{\mathcal{T}(x)}(v, y)$ where $\mathcal{T}(x)$ is the shortest path rooted at x in G.

Since $f \in sv$ and $f \in sy$, the edge $f \in sr$ where $r = \mathsf{LCA}_{\mathcal{T}(s)}(v, y)$. Thus the edge f does not belong to the path v-r-y in $\mathcal{T}(s)$. Hence there is an s-y path in $G - f$ of length $||sv \diamond f|| + ||vs|| + ||sy|| \leq ||sv \diamond f|| + ||vs|| + 4||sv||$ (by Claim 10).

- Thus $G - f$ has an s-y path of length $||sv \diamond f|| + 5||vs||$, plus a v-x path of length $||sv \diamond f|| + 2||vs||$. Since $||xy|| \leq 2||vs||$, the s-v distance estimate returned in this case is at most $2||sv \diamond f|| + 9||vs|| \leq 11||sv \diamond f||$.

7. $f \in vx$ and $f \in xy$ and $f \notin ys$. As seen in case 6, there is a v-x path in $G - f$ of length $||sv \diamond f|| + 2||vs||$. Moreover, since $f \in vx$ and $f \in xy$, the edge $f \in xw$ where $w = \mathsf{LCA}_{\mathcal{T}(x)}(v, y)$. Hence the edge f does not belong to the path v-w-y in $\mathcal{T}(x)$.

Thus there is a v-y path in $G - f$ of length at most $||vx|| + ||xy|| \leq 3||sv||$. Hence there is an x-v-y path of length at most $||sv \diamond f|| + 2||vs|| + 3||sv|| = ||sv \diamond f|| + 5||vs||$.

- So the s-v distance estimate returned in this case is at most $||sy|| + (||sv \diamond f|| + 5||vs||) + (||sv \diamond f|| + 2||vs||)$. Since $||sy|| \leq 4||sv||$, this is at most $2||sv \diamond f|| + 11||sv|| \leq 13||sv \diamond f||$.

8. $f \in vx$ and $f \in xy$ and $f \in ys$. As seen in case 7, there is a v-x path in $G - f$ of length $||sv \diamond f|| + 2||vs||$ and there is an x-y path of length at most $||sv \diamond f|| + 5||vs||$. It also follows from case 7 that there is a v-y path in $G - f$ of length at most $3||sv||$, thus there is an s-y path in $G - f$ of length at most $||sv \diamond f|| + 3||vs||$.

- So the s-v distance estimate returned in this case is at most $3||sv \diamond f|| + 10||sv|| \leq 13||sv \diamond f||$.

Thus the stretch of our approximate distance oracle is at most 13. We have already seen that the query answering time is $O(1)$. This finishes the proof of the lemma. ◀

Size of the oracle. We show below in Lemma 11 that the space taken up by the data structures constructed in all the steps of our algorithm is $\widetilde{O}(n^{4/3} + |S|n)$.

▶ **Lemma 11.** *The space needed to store all the data structures constructed by our algorithm is* $\widetilde{O}(n^{4/3} + |S|n)$.

Proof. The space taken up by the truncated shortest path trees $\hat{\mathcal{T}}(u)$ for all $u \in \mathcal{L}_1$ is $O(\sum_{v\in V} |vt_v|)$. Observe that $|vt_v| \le n^{1/3}$ (by Proposition 4). Thus $O(\sum_v |vt_v|) = O(n^{4/3})$. Similarly the space taken up by $\mathcal{T}(t)$ for all $t \in T_2$ is $O(n^{4/3}\log n + |S|n)$ since $|T_2| = |\mathcal{L}_2| + |S|$ and $|\mathcal{L}_2|$ is $O(n^{1/3}\log n)$. The size of the ST-oracle (where $T = T_2$) is also $\widetilde{O}(n^{4/3} + |S|n)$ (by Theorem 5).

For each vertex v, we store t_v and t'_v – these are the nearest vertices to v in T_1 and T_2, respectively. For all edges $f \in vt_v$, we store $||vt_v \diamond f||$ in the data structure Dist_1. We have $|vt_v| \le n^{1/3}$ (by Proposition 4). Thus the space taken by the data structure Dist_1 to store the distances $\mathsf{Dist}_1[v, i]$ where $v \in V$ and $1 \le i \le n^{1/3}$ is at most $n^{4/3}$.

For all edges $f \in ut'_u$, where $u \in T_1$, the data structure Dist_2 stores $||ut'_u \diamond f||$. For any vertex u, we have $|ut'_u| \le n^{2/3}$ (by Proposition 4). Thus the space taken by Dist_2 to store the distances $\mathsf{Dist}_2[u, i]$ where $u \in T_1$ and $1 \le i \le n^{2/3}$ is $|T_1| \cdot n^{2/3} = O((n^{2/3}\log n + |S|) \cdot n^{2/3})$, which is $O(n^{4/3}\log n + |S|n^{2/3})$. Thus the entire space taken up by all the data structures is $\widetilde{O}(n^{4/3} + |S|n)$. ◀

It is easy to see that our algorithm runs in expected polynomial time. As mentioned at the end of Section 3, by blowing up the sizes of $\mathcal{L}_1$ and $\mathcal{L}_2$ by a factor of $\log n$, their construction can be made deterministic as stated in [9, Lemma 1]. Moreover, we can easily ensure that $L_2 \subseteq L_1$. Thus Theorem 2 follows. We restate it below for convenience.

▶ **Theorem 2.** *Let $G = (V, E)$ be an undirected graph on n vertices with positive edge weights. For any $S \subseteq V$, a fault-tolerant sourcewise approximate distance oracle with multiplicative stretch at most 13 and size $\widetilde{O}(|S|n + n^{4/3})$ can be constructed in polynomial time such that $Qu(s, v, f)$ where $(s, v) \in S \times V$ and $f \in E$ can be answered in constant time.*

As mentioned in Remark 8, along with every distance in Dist_1 and Dist_2, we could also store the corresponding replacement paths in 2-decomposable form. Thus along with the approximate s-v distance, the query answering algorithm can also return the approximate path between s and v as the union of 3 paths, each in 2-decomposable form.

5 Concluding Remarks

Fault-tolerant approximate distance oracles that maintain approximate distances for all pairs of vertices have been well-studied. Fault-tolerant single source and multiple source *exact* distance oracles have also been studied. As mentioned in [9], given a subset $S \subseteq V$, for the problem of storing $||sv \diamond f||$ where $(s, v) \in S \times V$ and $f \notin E$ is allowed[3] (this is interpreted the same as if no edge has failed), using standard tools, it can be shown that there are n-vertex graph families, for which any representation that allows for the return of all the $S \times V$ post-failure distances must have size $\Omega(n^{3/2}\sqrt{|S|})$. This motivates the study of sparser data structures that maintain *approximate* distances for all pairs in $S \times V$ under the failure of any $f \in E$. Such a data structure is a fault-tolerant sourcewise approximate distance oracle.

We showed two such oracles: one of size $\widetilde{O}(|S|n + n^{3/2})$ and stretch 5 and another of size $\widetilde{O}(|S|n + n^{4/3})$ and stretch 13. The query time for both oracles is constant. Upon query $Qu(s, v, f)$ where $f \notin E$, it turns out that both our query answering algorithms return the original distance $||sv||$ as if no edge has failed. There are several interesting open problems:

- Are there approximate sourcewise distance oracles of size $\widetilde{O}(|S|n + n^{1+1/k})$ and stretch $8k - 3$ with $O(1)$ query answering time for all integers $k \ge 1$? Our constructions showed such oracles for $k = 1, 2$.

[3] We thank a reviewer for pointing out this subtlety to us.

- The study of fault-tolerant exact as well as approximate distance oracles has so far considered structured subsets of $V \times V$ such as $S \times T$. Is there a sparse fault-tolerant exact or approximate distance oracle for an arbitrary subset $\mathcal{P}$ of $V \times V$?

References

1 Yehuda Afek, Anat Bremler-Barr, Haim Kaplan, Edith Cohen, and Michael Merritt. Restoration by path concatenation: fast recovery of mpls paths. *Distributed Computing*, 15(4):273–283, 2002. doi:10.1007/s00446-002-0080-6.

2 Surender Baswana and Neelesh Khanna. Approximate shortest paths avoiding a failed vertex: Near optimal data structures for undirected unweighted graphs. *Algorithmica*, 66(1):18–50, 2013. doi:10.1007/S00453-012-9621-Y.

3 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.

4 Aaron Bernstein and David R. Karger. Improved distance sensitivity oracles via random sampling. In Shang-Hua Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 34–43. SIAM, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347087.

5 Aaron Bernstein and David R. Karger. A nearly optimal oracle for avoiding failed vertices and edges. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 101–110. ACM, 2009. doi:10.1145/1536414.1536431.

6 Davide Bilò, Shiri Chechik, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, Simon Krogmann, and Martin Schirneck. Approximate distance sensitivity oracles in subquadratic space. *TheoretiCS*, 3, 2024. doi:10.46298/THEORETICS.24.15.

7 Davide Bilò, Shiri Chechik, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, and Martin Schirneck. Improved distance (sensitivity) oracles with subquadratic space. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 1550–1558. IEEE, 2024. doi:10.1109/FOCS61266.2024.00097.

8 Davide Bilò, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, Simon Krogmann, and Martin Schirneck. Compact distance oracles with large sensitivity and low stretch. In Pat Morin and Subhash Suri, editors, *Algorithms and Data Structures - 18th International Symposium, WADS 2023, Montreal, QC, Canada, July 31 - August 2, 2023, Proceedings*, volume 14079 of *Lecture Notes in Computer Science*, pages 149–163. Springer, 2023. doi:10.1007/978-3-031-38906-1_11.

9 Davide Bilò, Keerti Choudhary, Luciano Gualà, Stefano Leucci, Merav Parter, and Guido Proietti. Efficient oracles and routing schemes for replacement paths. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 13:1–13:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.STACS.2018.13.

10 Davide Bilò, Sarel Cohen, Tobias Friedrich, and Martin Schirneck. Near-optimal deterministic single-source distance sensitivity oracles. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 18:1–18:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ESA.2021.18.

11 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Multiple-edge-fault-tolerant approximate shortest-path trees. *Algorithmica*, 84(1):37–59, 2022. URL: `https://doi.org/10.1007/s00453-021-00879-8`, `doi:10.1007/S00453-021-00879-8`.

12 Shiri Chechik, Sarel Cohen, Amos Fiat, and Haim Kaplan. $(1 + \epsilon)$-approximate f-sensitive distance oracles. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1479–1496. SIAM, 2017. `doi:10.1137/1.9781611974782.96`.

13 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. f-sensitivity distance oracles and routing schemes. *Algorithmica*, 63(4):861–882, 2012. `doi:10.1007/S00453-011-9543-0`.

14 Camil Demetrescu and Mikkel Thorup. Oracles for distances avoiding a link-failure. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 838–843, 2002. URL: `http://dl.acm.org/citation.cfm?id=545381.545490`.

15 Dipan Dey and Manoj Gupta. Near optimal algorithm for fault tolerant distance oracle and single source replacement path problem. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 42:1–42:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPICS.ESA.2022.42`.

16 Dipan Dey and Manoj Gupta. Near optimal dual fault tolerant distance oracle. In Timothy M. Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPIcs*, pages 45:1–45:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. `doi:10.4230/LIPICS.ESA.2024.45`.

17 Dipan Dey and Manoj Gupta. Nearly optimal fault tolerant distance oracle. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 944–955. ACM, 2024. `doi:10.1145/3618260.3649697`.

18 Ran Duan and Hanlin Ren. Maintaining exact distances under multiple edge failures. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1093–1101. ACM, 2022. `doi:10.1145/3519935.3520002`.

19 Ran Duan and Tianyi Zhang. Improved distance sensitivity oracles via tree partitioning. In Faith Ellen, Antonina Kolokolova, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures - 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017, Proceedings*, volume 10389 of *Lecture Notes in Computer Science*, pages 349–360. Springer, 2017. `doi:10.1007/978-3-319-62127-2_30`.

20 Manoj Gupta and Aditi Singh. Generic single edge fault tolerant exact distance oracle. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 72:1–72:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. `doi:10.4230/LIPICS.ICALP.2018.72`.

21 John Hershberger and Subhash Suri. Vickrey prices and shortest paths: What is an edge worth? In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 252–259. IEEE Computer Society, 2001. `doi:10.1109/SFCS.2001.959899`.

22 Merav Parter and David Peleg. Sparse fault-tolerant BFS trees. In Hans L. Bodlaender and Giuseppe F. Italiano, editors, *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, volume 8125 of *Lecture Notes in Computer Science*, pages 779–790. Springer, 2013. `doi:10.1007/978-3-642-40450-4_66`.