# Approximate Distance Oracles Subject to Multiple Vertex Failures*

Ran Duan †1, Yong Gu ‡1, and Hanlin Ren §1

[1]Institute for Interdisciplinary Information Sciences, Tsinghua University

December 29, 2020

**Abstract**

Given an undirected graph $G = (V, E)$ of $n$ vertices and $m$ edges with weights in $[1, W]$, we construct vertex sensitive distance oracles (VSDO), which are data structures that preprocess the graph, and answer the following kind of queries: Given a source vertex $u$, a target vertex $v$, and a batch of $d$ failed vertices $D$, output (an approximation of) the distance between $u$ and $v$ in $G - D$ (that is, the graph $G$ with vertices in $D$ removed). An oracle has stretch $\alpha$ if it always holds that $\delta_{G-D}(u, v) \le \tilde{\delta}(u, v, D) \le \alpha \cdot \delta_{G-D}(u, v)$, where $\delta_{G-D}(u, v)$ is the actual distance between $u$ and $v$ in $G - D$, and $\tilde{\delta}(u, v, D)$ is the distance reported by the oracle.

In this paper we construct efficient VSDOs for any number $d$ of failures. For any constant $c \ge 1$, we propose two oracles:

- The first oracle has size $n^{2+1/c}(\log n/\epsilon)^{O(d)} \cdot \log W$, answers a query in $\mathrm{poly}(\log n, d^c, \log\log W, \epsilon^{-1})$ time, and has stretch $1 + \epsilon$, for any constant $\epsilon > 0$.
- The second oracle has size $n^{2+1/c}\mathrm{poly}(\log(nW), d)$, answers a query in $\mathrm{poly}(\log n, d^c, \log\log W)$ time, and has stretch $\mathrm{poly}(\log n, d)$.

Both of these oracles can be preprocessed in time polynomial in their space complexity. These results are the first approximate distance oracles of poly-logarithmic query time for any constant number of vertex failures in general undirected graphs. Previously there are $(1+\epsilon)$-approximate *d-edge* sensitive distance oracles [Chechik et al. 2017] answering distance queries when $d$ edges fail, which have size $O(n^2(\log n/\epsilon)^d \cdot d \log W)$ and query time $\mathrm{poly}(\log n, d, \log\log W)$.

*This work has been supported in part by the Zhongguancun Haihua Institute for Frontier Information Technology.

†duanran@mail.tsinghua.edu.cn.

‡guyong12@mails.tsinghua.edu.cn.

§rhl16@mails.tsinghua.edu.cn.

# 1 Introduction

Real-life networks are prone to failures. Usually, there can be several failed nodes or links, but the graph topology will not deviate too much from the underlying failure-free graph. A typical problem is to find the shortest path between two nodes in a network that avoids a specific set of failed nodes or links. This motivates the *d-failure* model, in which we should preprocess a graph, such that upon a small number ($d$) of failures, we can "recover" from these failures quickly.

In their pioneering work, Demetrescu and Thorup [32] designed a data structure that can maintain all-pairs shortest paths under one edge failure. In other words, for each triple $(u, v, f)$ where $u, v$ are vertices and $f$ is a failed edge, the data structure can output the length of the shortest path from $u$ to $v$ that does not go through $f$, in $O(\log n)$ query time. A subsequent work [33] extends the structure to also handle one vertex failure, and improves the query time to $O(1)$. The one-failure case is studied extensively in literature [9, 12–14, 24, 28, 38, 41, 42, 58, 70].

People also tried to find structures handling multiple failures. For undirected graphs, we can answer connectivity queries under $d$ edge failures[1] [36, 37, 57] and $d$ vertex failures [36, 37] in $\text{poly}(d, \log n)$ time. Chechik et al. [26] designed a data structure that maintains $O(d)$-approximate shortest paths under $d$ edge failures in an undirected graph, and Bilò et al. [15] improved the approximation ratio to $2d + 1$. For any $\epsilon > 0$, Chechik et al. [25] designed a data structure that $(1 + \epsilon)$-approximates shortest paths under $d$ edge failures in an undirected graph, with space complexity $O(n^2(\log n/\epsilon)^d \cdot d \log W)$ and query time $\text{poly}(\log n, d, \log\log W)$, where $W$ is the ratio of the largest edge weight to the smallest edge weight. More related work can be found in Section 1.3.

However, despite much effort, it was not known if one can maintain (approximate) shortest paths under multiple *vertex* failures. This problem was addressed as an open problem in [9, 25, 26], and also in Chechik's PhD thesis [23].

In this paper we build efficient data structures that answer approximate distance queries under multiple vertex failures for general undirected graphs, answering the above question in the affirmative. A *vertex-sensitive distance oracle* (VSDO) for a weighted undirected graph $G = (V, E)$ is a data structure that given a set of failed vertices $D \subseteq V$ and $u, v \in V \setminus D$, outputs (an estimate of) the length of the shortest path from $u$ to $v$ that avoids all vertices in $D$. We assume a known upper bound $d$ on the number of failures, i.e. for any query $(u, v, D)$, we always have $|D| \leq d$. We will be concerned with the following parameters of a VSDO:

- Space complexity, i.e. the amount of space that the data structure occupies.
- Query time, i.e. the time needed to answer one query $(u, v, D)$.
- Approximation ratio, a.k.a. stretch: A VSDO has stretch $\alpha$ if it always holds that $\delta_{G-D}(u, v) \leq \tilde{\delta}(u, v, D) \leq \alpha \cdot \delta_{G-D}(u, v)$, where $\delta_{G-D}(u, v)$ is the actual distance between $u$ and $v$ in $G - D$ (i.e. $G$ with $D$ disabled), and $\tilde{\delta}(u, v, D)$ is the output of the VSDO.

We will not be particularly interested in the preprocessing time of VSDOs; nevertheless, all VSDOs in this paper can be preprocessed in time polynomial in their space complexity.

In this paper, $n$ and $m$ denote the number of vertices and edges respectively. Let $W$ be the ratio of the largest edge weight to the smallest edge weight. W.l.o.g. we can assume that edge weights are real numbers in $[1, W]$.

[1] We can also use dynamic connectivity structures with poly-logarithmic worst case update time [40, 47, 69] to handle $d$ edge failures.

## 1.1 Our Results

We provide the first constructions of approximate VSDOs for general undirected graphs with poly-logarithmic query time. Our main results are as follows:[2]

**Theorem 1.1** (main)**.** *For any constants $c \geq 1$ and $\epsilon > 0$, we can construct VSDOs for undirected graphs with:*

*(a) space complexity $n^{2+1/c} \log W \cdot (\epsilon^{-1} \log n)^{O(d)}$, query time $\tilde{O}(d^{2c+6}\epsilon^{-1} \log\log W)$ and stretch $1+\epsilon$;*
*(b) space complexity $\tilde{O}(n^{2+1/c}d^3 \log(nW))$, query time $\tilde{O}(d^{2c+9} \log\log(nW))$ and stretch $O(d^{c+2} \log^6 n)$.*

*Each oracle can be preprocessed in time polynomial in their space complexity.[3] Our constructions also allow an actual approximate shortest path to be retrieved in an additional time of $O(\ell)$, where $\ell$ is the number of edges in the reported path.*

Using existing structures, we need either $n^{\Omega(d)}$ space or $\Omega(n)$ query time.[4] Thus our results are the first of its kind.

## 1.2 A Brief Overview

In this section, we briefly introduce the ideas needed to construct the desired VSDOs.

**The edge-sensitive distance oracle of [25].** Our first VSDO depends on [25] which handles $d$ *edge* failures. Therefore we briefly describe their oracle first. It may be helpful to think of their query algorithm as a recursive one.

Given $u, v \in V$ and a set $D$ of $d$ edge failures, let $P_{\text{ans}}$ be the shortest $u$-$v$ path in $G - D$, which we are searching for. The oracle first partitions the shortest path $P$ from $u$ to $v$ in $G$ (which may go through failures) into $\tilde{O}(\epsilon^{-1} \log W)$ short segments. Consider a segment $X$ that contains some failed edges. If $P_{\text{ans}}$ does not go through $X$, then we can "preprocess" the graph $G - X$ and search for $P_{\text{ans}}$ in $G - X$. Otherwise, if $P_{\text{ans}}$ goes through some vertex $x \in X$, then we can pick an *arbitrary* vertex $w \in X$ such that there are no failed edges between $x$ and $w$, and *pretend* that $P_{\text{ans}}$ passes through $w$. That is, we recursively find the shortest paths in $G - D$ from $u$ to $w$ and from $w$ to $v$ and concatenate them. It is easy to see that this brings an additive error of at most $2|X|$ to our solution, where $|X|$ is the length of $X$.

Thus, we want to find a small set of intermediate vertices, which we denote as $H$, with the following property: For every vertex $x$ and failure $f$, if $x$ has distance at most $|X|$ to $f$, then there is some $w \in H$ such that $x$ also has distance at most $|X|$ to $w$ in $G - D$. As it turns out that the query time is polynomial in $|H|$, the size of $H$ should be small.

There is a natural choice of $H$: we simply let it be the set of vertices incident to some failed edges. It is easy to see that $|H| \leq 2d$, thus the query algorithm runs in time $\text{poly}(d)$. The above property is also true: given any vertex $x$ and a nearby failure $f$, we can walk along the path from $x$ to $f$ until we meet a failed edge, then the vertex $w$ we stop at is both in $H$ and close to $x$. We can control the total additive error (i.e. the sum of $2|X|$'s over the "recursion") to be at most $\epsilon \cdot |P_{\text{ans}}|$, by partitioning each path into *sufficiently short* segments.

[2] $\tilde{O}$ hides $\text{poly}(\log n)$ factors.

[3] See Table 3 in Appendix B for precise time bounds.

[4] We can use the $d$-fault tolerant spanner [17,19,22,34] with the brute-force query algorithm, build $n^{d-2}$ two-failure distance oracles [35], use the dynamic shortest path algorithms [67], or use the oracle [68] which also works for directed graphs. But none of these solutions provide both $n^{o(d)}$ space and $o(n)$ query time.

Note that, for the sake of intuition, we have omitted some important details, such as how to "preprocess" $G - X$ (by a decision tree structure) and how to implement the query algorithm (non-recursively).

**The "high-degree" obstacle.** The obvious difficulty of handling vertex failures is the presence of failed vertices with very high degrees. If every failed vertex has degree $\leq \Delta$, we can simply simulate an edge-failure distance oracle [25, 26] and delete at most $d \cdot \Delta$ edges from it. Equivalently, we can define the set of intermediate vertices $H$ as those non-failure vertices adjacent to some failure, then $|H| \leq d \cdot \Delta$ and we run the above query algorithm. However the techniques of [25, 26] do not seem to work for high-degree vertex failures. For example, techniques in [26] only guarantee a stretch of $\geq \Delta$, and techniques in [25] require poly($\Delta$) query time, therefore both are unsatisfactory when $\Delta = \Omega(n)$.

By the construction of $(2k-1)$-stretch spanners with $O(n^{1+1/k})$ edges [8], we can construct a $(2\log n - 1)$-stretch spanner with $O(n)$ edges. We note that the query algorithm works even if every failed vertex has a small degree in the spanner (rather than in the whole graph): We can define $H$ to be the set of vertices adjacent to some failed vertex *in the spanner*. If $P_{\text{ans}}$ goes through some vertex $x$ that has distance $|X|$ to a failed vertex $f$, the distance between $x$ and $f$ in the spanner is $O(|X| \log n)$, and there must be some $w \in H$ that has distance $O(|X| \log n)$ to $x$ in $G - D$. By partitioning the paths into shorter segments, we can still control the additive error, i.e. the sum of $O(|X| \log n)$ over the "recursion", to be less than $\epsilon \cdot |P_{\text{ans}}|$.

**High-degree hierarchy: A first attempt.** Given the "high-degree" obstacle, it is natural to see whether the "high-degree hierarchy" of [36] may help us. Plugging the spanners[5] into the hierarchy of [36], we obtain a structure as follows. The vertices are partitioned into $p = O(\log n)$ levels; let $U_i$ be the set of vertices with level $\geq i$. So we have a sequence of vertex sets $V = U_1 \supseteq U_2 \supseteq \cdots \supseteq U_p \supseteq U_{p+1} = \emptyset$, and the $i$-th level is the set $U_i \setminus U_{i+1}$.[6] For every $i$, let $G_i$ be the induced subgraph of $V \setminus U_{i+1}$. We do not have a complete spanner for $G_i$; we can only afford to build a "subset-spanner" that preserves the distances in $G_i$, among vertices in $U_i \setminus U_{i+1}$ (instead of $V \setminus U_{i+1}$). The structure guarantees that every failed vertex in the subset-spanner of any level has low degrees.

It is natural to define $H$ as the set of neighbors of failures in the subset-spanners, and $|H|$ will be small. If $P_{\text{ans}}$ goes through some vertex $x$ that has distance $|X|$ to a failed vertex $f$, and $x$ *and* $f$ *are in the same level*, then we can find an intermediate vertex $w \in H$ that has distance $O(|X| \log n)$ to $x$ in $G - D$, and we are fine. But what if $x$ and $f$ are in different levels? In this case, the $x$-$f$ path may not be preserved by the "*subset*-spanner", thus not captured by $H$. In [36, Section 4], the authors used ad hoc structures to preserve connectivity between different levels; it appears difficult to extend these structures to also handle ($(1+\epsilon)$-approximate) distances.

**Our ideas.** It is inconvenient that the spanner at level $i$ only preserves distances inside $U_i \setminus U_{i+1}$. Therefore, our first idea is to "extend" the spanners to also preserve distances at lower levels: the spanner at level $i$ should preserve distances between any pair of vertices $(x, y)$, where $x \in U_i \setminus U_{i+1}$ and $y \in V \setminus U_{i+1}$. Note that we still only guarantee that every vertex failure has small degrees in the *original* spanners; they may have large degrees in the extended spanners.

We implement the spanners by *tree covers*, and there is a natural way to "extend" them. The extended tree cover consists of a collection of trees whose union is a spanner that preserves distances

[5] The reason that we need to plug in a spanner, rather than the original graph, is that we can only plug in a *sparse* graph into the high-degree hierarchy.

[6] In the hierarchy structure of Section 2.2, each $U_{i+1}$ is not necessarily a subset of $U_i$; this issue is not essential, so for simplicity, in the brief overview we will assume each $U_{i+1}$ is indeed a subset of $U_i$.

between $U_i \setminus U_{i+1}$ and $V \setminus U_{i+1}$. Moreover, each tree is a shortest path tree rooted in $U_i \setminus U_{i+1}$ (the highest level of $G_i$). See Section 2.1 for more details.

Recall that in the query algorithm, we have a non-failure vertex $x$ that is close to a failure $f$, and we want to find an intermediate vertex $w \in H$ that is close to $x$ in $G - D$. Suppose that $x$ is at a higher level than $f$. If we walk from $f$ (at a lower level) to $x$ (at a higher level), it seems that our first step should go to the parent of $f$ in some tree. Actually, this intuition can be rigorously proved! See the proof of Lemma 3.5. Therefore, if $H$ consists of the neighbors of every failure (in the original spanners) and the parents of every failure in each tree (in the extended tree covers), then we can deal with every $(x, f)$ such that the level of $x$ is at least that of $f$. Every failure is only in $\tilde{O}(1)$ trees, thus $|H|$ is indeed small.

We need to adapt the query algorithm to ensure that $f$ never has a higher level than $x$. Let $P$ be the shortest $u$-$v$ path in the original graph, and we partition $P$ into short segments. Consider a segment $X$ that contains failures, and let $i$ be the highest level of any failure in $X$. If $P_{\text{ans}}$ does not contain any vertex in $X$ *with level at least* $i$, then we can "preprocess" the graph $G - (X \cap U_i)$ and search for $P_{\text{ans}}$ in this subgraph. Otherwise $P_{\text{ans}}$ goes through some $x \in (X \cap U_i)$, and by definition, the level of $x$ cannot be smaller than the level of any failure in $X$. Therefore, we can find some intermediate vertex $w \in H$ close to $x$, "pretend" that $P_{\text{ans}}$ goes through $w$, and continue.

The above discussion implies a data structure with space complexity roughly $n^3$. To reduce the space complexity by a factor of $n^{1-o(1)}$, we prove a structural theorem (Theorem 4.3) for shortest paths under vertex failures, which allows us to compress such paths. (The corresponding theorem [25, Theorem 3.1] does not hold for vertex failures.) Curiously, the proof of this theorem also relies on Lemma 3.5.

**On oracle (b).** Although oracle (b) has a larger stretch compared to oracle (a), we think it is also of interest, since it is the first oracle that handles $\omega(\log n)$ failures in polynomial space and $\text{poly}(\log n)$ query time, within a reasonable stretch.[7] Note that setting $\epsilon = \omega(1)$ (e.g. $\epsilon = \log n$) in oracle (a) does *not* improve its space complexity to $n^2 \log^{o(d)} n$, so oracle (b) is *not* a direct corollary of oracle (a).

### 1.3 More Related Work

**Sensitivity oracles.** For the case of two vertex failures, Duan and Pettie [35] showed that exact distances in a directed weighted graph can be queried in $O(\log n)$ time, with an oracle of size $O(n^2 \log^3 n)$, and Choudhary [27] designed an oracle of $O(n)$ size that handles single source reachability queries in directed graphs in $O(1)$ time.

The general problem of $d$ failures has also received attention on *planar* graphs: Borradaile et al. [20] constructed a data structure that maintains connectivity under $d$ vertex failures, and Charalampopoulos et al. [21] designed a data structure that answers exact distance queries under $d$ vertex failures.

In a recent breakthrough, van den Brand and Saranurak [68] gave an oracle that handles an arbitrary number $d$ of edge failures in *directed* graphs. Their oracle can answer reachability queries in $O(d^\omega)$ time, and exact distance queries in $n^{2-\Omega(1)}$ time (for small integer weights), where $\omega < 2.3728639$ is the matrix-multiplication exponent [29, 50, 63, 71].

We summarize the sensitivity connectivity/distance oracles in Table 2 of Appendix B.

[7] It seems that even $O(\sqrt{n})$ stretch was open before this result.

**Fault-tolerant structures.** A related concept is *fault-tolerant (FT) spanners*: a subgraph $G'$ of $G$ is a $d$-FT spanner if, after removing any $d$ vertices, the remaining parts of $G'$ is a spanner of the remaining parts of $G$. It might be *a priori* surprising that sparse FT spanners exist, but Chechik et al. [22] gave the first construction of $d$-FT $(2k-1)$-spanners with $O(d^2k^{d+1} \cdot n^{1+1/k} \log^{1-1/k} n)$ edges. Subsequent papers [17, 19, 34] improved the number of edges to $O(n^{1+1/k}d^{1-1/k})$, which is optimal assuming the girth conjecture of Erdős [39].

Besides FT spanners, there are many other kinds of fault-tolerant structures, e.g. [15, 16, 18, 51, 53–56]. We refer the reader to the excellent survey of [52].

**Dynamic shortest path.** There are dynamic all-pairs shortest path structures handling vertex updates. Thorup [65] gave a fully dynamic all-pairs shortest paths structure with worst-case update time $\tilde{O}(n^{2.75})$, and Abraham et al. [1] gave a randomized worst-case update time bound $\tilde{O}(n^{2+2/3})$. Recently, Brand and Nanongkai [67] gave a $(1+\epsilon)$-approximate algorithm for maintaining APSP under edge insertions and deletions with worst-case update time $\tilde{O}(n^{1.863}/\epsilon^2)$ for directed graphs. Other fully or partial dynamic shortest path structures include [2, 11, 31, 44–46, 49, 60–62, 64].

### 1.4 Notation

In this paper, $\log x = \log_2 x$, and $\ln x = \log_e x$. For a set $S$ and an integer $k$, $|S|$ is the cardinality of $S$, and we denote $\binom{S}{k} = \{S' \subseteq S : |S'| = k\}$, and $\binom{S}{\le k}, \binom{S}{\ge k}$ are defined analogously. For two sets $X$ and $Y$, define their Cartesian product as $X \times Y = \{(x, y) : x \in X, y \in Y\}$. We use $\circ$ as the concatenation operator for paths or sequences. For paths $P_1, P_2$, if $u$ is the last vertex in $P_1$ and $v$ is the first vertex in $P_2$, then $P_1 \circ P_2$ is well-defined if $u = v$ or $(u, v)$ is an edge in $G$.

For a graph $H$ and $u, v \in V(H)$, $w_H(u, v)$ denotes the length of the edge between $u$ and $v$ ($w_H(u, v) = +\infty$ if such an edge does not exist), $\delta_H(u, v)$ denotes the length of the shortest path in $H$ from $u$ to $v$ and $\pi_H(u, v)$ denotes the corresponding shortest path. If $S \subseteq V(H)$ is a subset of vertices, then $\delta_H(u, S) = \min\{\delta_H(u, v) : v \in S\}$. ($\delta_H(u, \emptyset) = +\infty$.) We omit the subscript $H$ if $H = G$ is the input graph. We define $H[S]$ as the subgraph induced by $S$, and $H - S = H[V(H) \setminus S]$. We use $nW$ as an upper bound of the diameter of any (connected) subgraph of $G$. We assume that the shortest path between every pair of vertices in any subgraph is unique (see Section 3.4 of [30]).

For a path $P$ and $u, v \in P$, define $P[u, v]$ as the portion from $u$ to $v$ in $P$, and sometimes this notation emphasizes the *direction* from $u$ to $v$. Let $(u = x_0, x_1, \ldots, x_{\ell-1}, x_\ell = v)$ denote the path $P[u, v]$, then we define $P(u, v] = P[x_1, v], P[u, v) = P[u, x_{\ell-1}]$ and $P(u, v) = P[x_1, x_{\ell-1}]$. Define $|P|$ as the length of path $P$. For a tree $T$ rooted at $r$ and a vertex $x \in V$, define the depth of $x$, denoted by $\mathrm{dep}_T(x)$, as the (weighted) distance from $x$ to $r$ in $T$.

In this paper, $D$ denotes the set of $\le d$ failed vertices. For convenience, we always assume $n \ge 3$ and $d \ge 2$.

Note that we also define some more notations at the end of Section 2.2, which is relevant to the "high-degree hierarchy". Table 1 in Appendix B summarizes some nonstandard notation in this paper.

## 2 Source-Restricted Tree Covers in High-Degree Hierarchy

Our VSDO is based on a variant of the *high-degree hierarchy* of [36], which we equip with the *source-restricted tree covers* of [59, 66] to approximately preserve distances.

## 2.1 Source-Restricted Tree Covers

Let $G = (V, E)$ be an undirected graph. A *tree cover* of $G$ is, informally, a set of trees such that every vertex $v \in V$ is in a small number of trees, and for every two vertices $u, v \in V$, there is a tree that approximately preserves their distance $\delta(u, v)$. In this paper, we relax the second condition, requiring it to hold only for every $u \in S, v \in V$, where $S$ is some subset of $V$. Following terminologies of [59], we call such tree covers *source-restricted*.

Throughout this paper, $k = \ln n$.[8] We define *source-restricted tree cover* as follows.

**Definition 2.1.** Given $S \subseteq V$, an *$S$-restricted tree cover* is a set of rooted trees $\{T(w) : w \in S\}$, such that the following hold.

a) For every $w \in S$, there is exactly one tree $T(w)$ rooted at $w$, spanning a subset of $V$ (which we denote as $V(T(w))$).
b) For every $u \in S, v \in V$, there is some $w \in S$ such that $u, v \in V(T(w))$, and $\mathrm{dep}_{T(w)}(u) + \mathrm{dep}_{T(w)}(v) \le (2k-1)\delta(u, v)$.[9]
c) Every vertex $v \in V$ is in at most $kn^{1/k}(\ln n + 1) \le 2e \ln^2 n$ trees.

In [66], Thorup and Zwick constructed approximate distance oracles, and they noticed that their constructions are also good tree covers. A simple modification of their construction (see [59]) yields source-restricted tree covers.

**Theorem 2.2.** *Given a graph $G = (V, E)$ and $S \subseteq V$, we can compute in deterministic polynomial time an $S$-restricted tree cover $\mathcal{T}(S) = \{T(w) : w \in S\}$ such that for any $u \in S, v \in V$, the vertex $w$ in Definition 2.1 b) can be found in $O(k)$ time.*

For completeness, we provide a sketch of the construction in Appendix A; we also refer the interested reader to [59, 66] for details of this construction.

For $S \subseteq V$, we denote $\mathcal{T}(S)$ as the $S$-restricted tree cover constructed in Theorem 2.2. For $S, R \subseteq V$, we denote $\mathcal{T}_R(S)$ as the $(S \setminus R)$-restricted tree cover $\mathcal{T}(S \setminus R)$ in $G - R$.

For technical reasons (namely, we want the hierarchy structure in Section 2.2 to have a reasonable size), we need that the number of "high-degree" vertices in $\mathcal{T}(S)$ is only $o(|S|/d)$, where $d$ is the number of failures. However, here we defined the tree cover $\mathcal{T}(S)$ to span not only $S$, but maybe some other vertices in $V$.[10] So we can only prove degree bounds of the following form: the number of vertices with high degree w.r.t. the "trunk" parts of the tree cover is $o(|S|/d)$. The precise definitions are as follows.

**Definition 2.3.** Consider $S \subseteq V$, $T \in \mathcal{T}(S)$, $v \in V(T)$. We say $v$ is a *trunk vertex* of $T$ if there are $u, w \in S$ such that $v$ lies on the path from $u$ to $w$ in $T$. The subtree (subgraph) of $T$ induced by trunk vertices of $T$ is denoted as $\mathrm{Trunk}(T)$. The *pseudo-degree* of a vertex $v \in V(T)$, denoted as $\mathrm{pdeg}_T(v)$, is the degree of $v$ in $\mathrm{Trunk}(T)$. If $v$ is not a trunk vertex of $T$, then $\mathrm{pdeg}_T(v) = 0$.

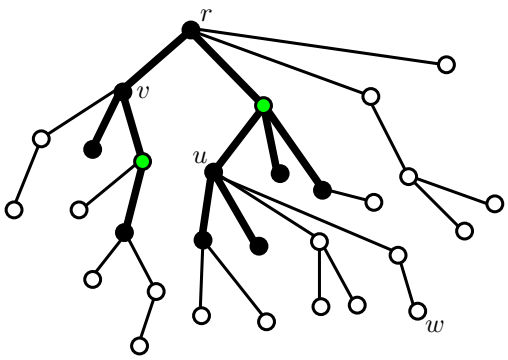
Note that vertices in $\mathrm{Trunk}(T)$ are not necessarily in $S$. See Fig. 1 as an example.

The following property will be useful in Section 3: for a vertex $v$ that is not in $\mathrm{Trunk}(T)$, its path in $T$ to any vertex in $S$ must go through its parent. (This is because the root of $T$ is always in $S$.)

[8] Our construction works for any parameter $k$, but the complexity is proportional to $kn^{1/k}$, so we minimize it by setting $k = \ln n$.

[9] We only require that the *distance* between $u$ and $v$ in $T(w)$ approximates $\delta(u, v)$ well in Section 3. However, we will require that the *sum of depths* of $u$ and $v$ approximates $\delta(u, v)$ well in Section 5.

[10] This corresponds to the informal description of "extending" tree covers in Section 1.2.




| | degree | pseudo-degree |
|---|---|---|
| $r$ | 4 | 2 |
| $u$ | 5 | 3 |
| $v$ | 4 | 3 |
| $w$ | 1 | 0 |

Figure 1: A sample tree in $\mathcal{T}(S)$. Black vertices are in $S$, green vertices are trunk vertices not in $S$, and bold edges denote the subtree induced by trunk vertices. We also include a table of degrees and pseudo-degrees of some sample vertices.

Let $s = 4e \cdot d^{c+1} \ln^2 n + 1$ be a degree threshold, where $c \geq 1$ is any constant. Define $\mathrm{Hi}(\mathcal{T}(S))$ as the set of vertices in $V$ that has pseudo-degree $> s$ in some tree in $\mathcal{T}(S)$. We prove our desired upper bound on $|\mathrm{Hi}(\mathcal{T}(S))|$.

**Lemma 2.4.** *For any $S \subseteq V$, $|\mathrm{Hi}(\mathcal{T}(S))| \leq \frac{|S|}{2d^{c+1}}$.*

*Proof.* For a tree $T$, let $\mathrm{Leaf}(T)$ be the set of leaves of $T$. Then there are at most $\left\lfloor \frac{|\mathrm{Leaf}(T)|-2}{s-1} \right\rfloor$ vertices in $T$ that has degree $> s$ [36, Lemma 3.1]. For any $T \in \mathcal{T}(S)$, $\mathrm{Leaf}(\mathrm{Trunk}(T)) \subseteq S$ by definition, thus

$$\sum_{T \in \mathcal{T}(S)} |\mathrm{Leaf}(\mathrm{Trunk}(T))| \leq \sum_{v \in S} |\{T \in \mathcal{T}(S) : v \in T\}| .$$

Since every $v \in S$ appears in $\leq 2e \ln^2 n$ trees in $\mathcal{T}(S)$, we have

$$\sum_{T \in \mathcal{T}(S)} |\mathrm{Leaf}(\mathrm{Trunk}(T))| \leq |S| \cdot 2e \ln^2 n,$$

thus

$$|\mathrm{Hi}(\mathcal{T}(S))| \leq \sum_{T \in \mathcal{T}(S)} \left\lfloor \frac{|\mathrm{Leaf}(\mathrm{Trunk}(T))|}{s-1} \right\rfloor \leq \frac{|S| \cdot 2e \ln^2 n}{s-1} = \frac{|S|}{2d^{c+1}}.$$ □

## 2.2 The High-Degree Hierarchy

We use a simplified version of the high-degree hierarchy in [36]. Fix a parameter $c \geq 1$, the hierarchy structure is a set of $O(n^{1/c})$ representations of the graph, such that for every set of $d$ failures, we can find some representation in which all failed vertices have low pseudo-degrees in their relevant tree covers.

**Definition 2.5.** The *hierarchy tree* is a rooted tree in which every node[11] corresponds to a subset of $V$. The root corresponds to $V$. Each node $U$ ($U \subseteq V$) stores a tree cover $\mathcal{T}(U)$, and each edge $(U, W)$, where $U$ is the parent of $W$, stores a tree cover $\mathcal{T}_W(U)$, which is $\mathcal{T}(U \setminus W)$ in $G - W$. The

[11] We use "vertex" for nodes in the input graph, and "node" for nodes in the hierarchy tree.

**Algorithm 1** Path finding algorithm

$U_1 \leftarrow V$
**for** $i \leftarrow 1$ to $h$ **do**
    If $U_i$ is a leaf then set $p \leftarrow i$ and halt
    Let $W_1, W_2, \ldots, W_d$ be the children of $U_i$ and artificially define $W_0 = \emptyset$, $W_{d+1} = W_d$.
    Let $j \in [0, d]$ be minimal such that $D \cap (W_{j+1} \setminus W_j) = \emptyset$
    If $j = 0$ then set $p \leftarrow i$ and halt
    Otherwise $U_{i+1} \leftarrow W_j$

hierarchy tree is constructed as follows. Let $U$ be any node. If $\mathrm{Hi}(\mathcal{T}(U)) = \emptyset$, then $U$ is a leaf; otherwise let $W_1, W_2, \ldots, W_d$ be its children, where

$$\begin{aligned} W_1 &= \mathrm{Hi}(\mathcal{T}(U)), && (1)\\ W_i &= W_{i-1} \cup \mathrm{Hi}(\mathcal{T}_{W_{i-1}}(U)). && (\text{for } 2 \le i \le d) \end{aligned}$$

Then we recursively deal with all $W_1, W_2, \ldots, W_d$.

There are two main differences compared with the hierarchy structure in [36].

- We simplified the definition of $W_1$ as in (1). This change is not essential, but we feel that it could make the hierarchy tree easier to understand. As a consequence, a node is not necessarily a subset of its parent, which is different from [36] (and Section 1.2).
- More importantly, we store in each node the source-restricted tree covers introduced in Section 2.1. By contrast, [36] only concerns about connectivity, so they used a (somewhat arbitrary) spanning forest instead.

The following lemmas assert that the hierarchy tree "is small, shallow and effectively represents the graph" [36], which are crucial for our data structures.

**Lemma 2.6** (Hierarchy Size and Depth)**.** *Consider the hierarchy tree constructed with high-degree threshold $s = 4e \cdot d^{c+1} \ln^2 n + 1$, then the following hold.*

1. *The depth $h$ of the hierarchy tree is at most $\lfloor \frac{1}{c} \log_d n \rfloor$,*[12] *assuming the root has depth 0.*
2. *The number of nodes in the hierarchy tree is at most $O(n^{1/c})$.*

*Proof.* Let $U$ be a node in the hierarchy tree, $W_1, W_2, \ldots, W_d$ be its children (if exist). By Lemma 2.4, we have $|W_1| = |\mathrm{Hi}(\mathcal{T}(U))| \le \frac{|U|}{2d^{c+1}}$ and $|\mathrm{Hi}(\mathcal{T}_{W_i}(U))| \le \frac{|U|}{2d^{c+1}}$ for any $0 < i \le d$. Since $W_{i+1} = W_i \cup \mathrm{Hi}(\mathcal{T}_{W_i}(U))$, it follows that $|W_i| \le i|U|/2d^{c+1}$ for each $i$. Therefore $|W_i| \le |U|/2d^c$ for all $0 < i \le d$. Any node at the $k$-th level corresponds to a subset of $V$ with size at most $n/(2d^c)^k$, therefore the depth of the hierarchy tree is at most $h \le \lfloor \log_{2d^c} n \rfloor \le \lfloor \frac{1}{c} \log_d n \rfloor$. There are at most $\sum_{i=0}^{h} d^i \le 2d^h = O(n^{1/c})$ nodes in the hierarchy tree. □

**Lemma 2.7.** *For any set $D$ of at most $d$ failures, Algorithm 1 finds in $O(hd)$ time a path $U_1(= V), U_2, \ldots, U_p$ in the hierarchy tree from root to some node $U_p$, such that for every tree $T \in \bigcup_{1 \le i \le p} \mathcal{T}_{U_{i+1}}(U_i)$ and every $f \in D$, $f$ has pseudo-degree $\le s$ in $T$. (Assume $U_{p+1} = \emptyset$.)*

[12] In subsequent sections we will write $h$ as a shorthand of $O(\log n / \log d)$ in time/space bounds.

*Proof.* Algorithm 1 executes at most $O(h)$ iterations since the hierarchy tree has depth at most $h$. For every node $U$ and vertex $v \in V$, we store the first child $W_j$ of $U$ (or none) that $v$ appears in. It is then easy to implement each iteration in $O(d)$ time. When Algorithm 1 halts at Line 3 or 6, either $U_p$ is a leaf or $D \cap W_1 = \emptyset$, where $W_1 = \mathrm{Hi}(\mathcal{T}(U_p))$ is the first child of $U_p$. Clearly, in both case we have $D \cap \mathrm{Hi}(\mathcal{T}(U_p)) = \emptyset$.

For any $1 \le i < p$, since $W_{d+1} = W_d$, Line 5 can always find such $j$. Let $U_{i+1} = W_j$ be the $j$-th child of $U_i$. If $j = d$, then $D \cap (W_{j'+1} \setminus W_{j'}) \neq \emptyset$ for $j' = 0, \ldots, d-1$. Since $|D| \le d$, we have $D \subseteq W_d$, thus $D \cap \mathrm{Hi}(\mathcal{T}_{W_d}(U_i)) = \emptyset$. If $1 \le j < d$, then we have $D \cap (W_{j+1} \setminus W_j) = \emptyset$. Since $W_{j+1} = W_j \cup \mathrm{Hi}(\mathcal{T}_{W_j}(U_i))$ and $\mathrm{Hi}(\mathcal{T}_{W_j}(U_i)) \cap W_j = \emptyset$, we have $D \cap \mathrm{Hi}(\mathcal{T}_{W_j}(U_i)) = \emptyset$. The lemma follows. □

In Section 3, Section 4 and Section 5, we always deal with a path $V = U_1, U_2, \ldots, U_p$ in the hierarchy tree from the root $V$ to a node $U_p$ (not necessarily a leaf node). Artificially define $U_{p+1} = \emptyset$. In other words:

- We run the preprocessing algorithm for every possible such paths $V = U_1, U_2, \ldots, U_p$, and we build a separate data structure for each path. The space complexity is then multiplied by a factor of $O(n^{1/c})$.
- In the query algorithm, given $D$, we always begin by identifying a path $V = U_1, U_2, \ldots, U_p$ using Lemma 2.7, then every failed vertex $f \in D$ has pseudo-degree $\le s$ in every $\mathcal{T}_{U_{i+1}}(U_i)$ $(1 \le i \le p)$.

Fix a path $V = U_1, U_2, \ldots, U_p$ in the hierarchy tree, and we assume $U_{p+1} = \emptyset$. The following corollary of Theorem 2.2 will be important for us.

**Corollary 2.8.** *Let $x \in U_\ell \setminus U_{\ell+1}$ and $y \in V \setminus U_{\ell+1}$. There is a tree $T \in \mathcal{T}_{U_{\ell+1}}(U_\ell)$ such that*

$$\mathrm{dep}_T(x) + \mathrm{dep}_T(y) \le (2k-1)\delta_{G-U_{\ell+1}}(x, y).$$

*Moreover, the root of such a tree can be found in $O(k)$ time.*

We will denote this tree $T$ as $T_\ell(x, y)$, and denote the path from $x$ to $y$ in $T$ as $\mathcal{P}_\ell(x, y)$.

The set of trees is denoted as $\mathcal{T} = \bigcup_{i=1}^{p} \mathcal{T}_{U_{i+1}}(U_i)$. The proof of Lemma 2.6 shows that $|U_i| \le |U_{i-1}|/2d^c$, therefore $|\mathcal{T}| \le \sum_{i=1}^{p} |U_i| = O(n)$. In a path $V = U_1, U_2, \ldots, U_p$ in the hierarchy tree, since $U_{i+1}$ is not necessarily a subset of $U_i$, we define the *level* of a vertex $v$ as:

**Definition 2.9.** Fix a path $V = U_1, U_2, \ldots, U_p$ in the hierarchy tree, define the *level* of $v$ to be the largest integer $l$ such that $v \in U_l$, denoted as $l(v)$. Define $G_\ell$ to be the subgraph of $G$ induced by all vertices with level at most $\ell$.

# 3 An $(1+\epsilon)$-Stretch Oracle with $n^{3+1/c+o(1)}$ Space

In this section we present an oracle with

$$\begin{aligned} \text{space complexity} &\quad n^{3+1/c} \cdot (\epsilon^{-1} \log(nW))^{O(d)}, \\ \text{query complexity} &\quad \mathrm{poly}(\log(nW), \epsilon^{-1}, d), \\ \text{stretch} &\quad 1+\epsilon, \end{aligned}$$

for any $\epsilon > 0$. In this paper (except Section 5 and Section 4.5) we may assume $d = o\Big(\frac{\log n}{\log\log n}\Big)$, $W = \mathrm{poly}(n)$, so we can simplify the notation for space complexity to $n^{3+1/c+o(1)}$. We will show how to reduce the space complexity to $n^{2+1/c+o(1)}$ in Section 4.

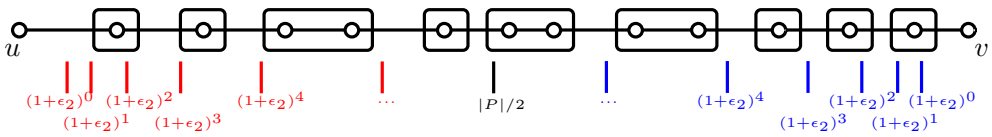



Figure 2: An illustration of path decomposition, where each rounded rectangle denotes a segment.

### 3.1 Data Structure

Let $\epsilon_1 = \epsilon/(2+\epsilon), \epsilon_2 = \epsilon_1/(2k-1)$, so $\epsilon_1 < 1$. (Recall $k = \ln n$.) We first define a decomposition of a path $P$ into $O(\log |P|/\epsilon_2)$ segments. This definition has the same spirit as [25, Definition 2.1], but it partitions the *vertices* (excluding $u, v$), rather than *edges*, into segments.

**Definition 3.1** ($(\epsilon_2$-)segments)**.** Consider a path $P = (u = v_0, v_1, v_2, \ldots, v_\ell = v)$. For every $1 \le i, j < \ell$, we say $v_i$ and $v_j$ are in the same *segment* if one of the two conditions hold:

- $|P[u, v_i]|, |P[u, v_j]| \le |P|/2$ and $\lfloor \log_{1+\epsilon_2} |P[u, v_i]| \rfloor = \lfloor \log_{1+\epsilon_2} |P[u, v_j]| \rfloor$.
- $|P[v_i, v]|, |P[v_j, v]| < |P|/2$ and $\lfloor \log_{1+\epsilon_2} |P[v_i, v]| \rfloor = \lfloor \log_{1+\epsilon_2} |P[v_j, v]| \rfloor$.

It is easy to verify that being in the same segment is indeed an equivalence relation, and each equivalence class is indeed a contiguous segment of the path. (See Fig. 2.) There are $O(\log |P|/\epsilon_2)$ different segments. For a vertex $v_i$ on $P$, define $\mathrm{seg}(v_i, P)$ as the segment it belongs to. More precisely, $\mathrm{seg}(v_i, P) = P[v_l, v_r]$ where $v_l$ is the leftmost (closest-to-$u$) vertex in the segment, and $v_r$ is the rightmost (closest-to-$v$) vertex in the segment. Define $\mathrm{seg}(P)$ as the set of segments on $P$. Note that $u$ and $v$ do not belong to any segment.

**Lemma 3.2.** *Let $P$ be a path from $u$ to $v$, $x \in P \backslash \{u, v\}$, then $|\mathrm{seg}(x, P)| \le \epsilon_2 \cdot \min\{|P[u, x]|, |P[x, v]|\}$.*

*Proof.* If $|P[u, x]| \le |P|/2$, $|P[u, x]| \le |P[x, v]|$ and $\mathrm{seg}(x, P)$ is defined in the first way. Let $i = \lfloor \log_{1+\epsilon_2} |P[u, x]| \rfloor$. For any $y \in \mathrm{seg}(x, P)$, $(1+\epsilon_2)^i \le |P[u, y]| < (1+\epsilon_2)^{i+1}$. Thus $|\mathrm{seg}(x, P)| \le (1+\epsilon_2)^{i+1} - (1+\epsilon_2)^i = \epsilon_2(1+\epsilon_2)^i \le \epsilon_2 |P[u, x]|$. The case that $|P[x, v]| < |P|/2$ is symmetric. □

Recall that we build a data structure for every node $U_p$ in the hierarchy tree. Let the path from the root $V$ to $U_p$ be $U_1 (= V), U_2 \ldots, U_p$ and $U_{p+1} = \emptyset$. The data structure for $U_p$ consists of decision trees $FT(u, v)$ for all pairs of vertices $u, v \in V$, which are constructed as follows:

- Each node[13] $\alpha \in FT(u, v)$ is associated with a set $\mathrm{avoid}(\alpha) \subseteq V$ of vertices that we avoid.
- Denote the root of $FT(u, v)$ as $\mathrm{root}(u, v)$, and let $\mathrm{avoid}(\mathrm{root}(u, v)) = \emptyset$.
- For each node $\alpha \in FT(u, v)$, we store the path $P_\alpha = \pi_{G - \mathrm{avoid}(\alpha)}(u, v)$, i.e. the shortest $u$-$v$ path in $G$ not passing through $\mathrm{avoid}(\alpha)$. If $u, v$ are not connected in $G - \mathrm{avoid}(\alpha)$, we assume that $P_\alpha$ is a path with length $+\infty$.
- For each node $\alpha$ of depth $< d$ (the root has depth 0), each segment $X \in \mathrm{seg}(P_\alpha)$, and each $1 \le i \le p$, we create a child $ch(\alpha, X, i)$ of $\alpha$, in which

$$\mathrm{avoid}(ch(\alpha, X, i)) = \mathrm{avoid}(\alpha) \cup (X \cap U_i). \tag{2}$$

  That is, the path stored in a child of $\alpha$ needs to avoid $\mathrm{avoid}(\alpha)$ and the vertices of $U_i$ in a segment $X \in \mathrm{seg}(P_\alpha)$.

[13] Recall that we use "vertex" for nodes in the input graph, and "node" for nodes in the decision trees and the hierarchy tree.

The decision tree has depth $d$, and each non-leaf node has $O(p \cdot \log |P|/\epsilon_2) = O(h\epsilon^{-1} \log n \log(nW))$ children. (Recall $h = O(\log n / \log d)$ and $\epsilon_2 = \epsilon/((2+\epsilon)(2k-1)) = \epsilon/\Theta(\log n)$.) We store in each node $\alpha$ the path $P_\alpha$ as well as a table of $\mathrm{seg}(v, P_\alpha)$ for each $v \in P_\alpha$, so that we can quickly locate any vertex in $P_\alpha$. Therefore one decision tree occupies $n \cdot O(h\epsilon^{-1} \log n \log(nW))^d$ space. As there are $O(n^{1/c})$ nodes in the hierarchy tree, and for each node we need to store $O(n^2)$ decision trees, the total space complexity is $n^{3+1/c} \cdot O(h\epsilon^{-1} \log n \log(nW))^d$.

### 3.2 Query Algorithm

Given $u, v$ and a set of failed vertices $D$, by Lemma 2.7 we first find a path $U_1 (= V), U_2 \ldots, U_p$ in the hierarchy tree, and set $U_{p+1} = \emptyset$. Let $\mathcal{T} = \bigcup_{i=1}^{p} \mathcal{T}_{U_{i+1}}(U_i)$, then by Lemma 2.7, the pseudo-degrees of all $f \in D$ in every tree in $\mathcal{T}$ is at most $s$.

As in [25], the query algorithm builds an auxiliary graph $H$, but the definition of $H$ is different from [25]. The query algorithm builds $H$ as Definition 3.3, and outputs $|\pi_H(u, v)|$ as an $(1+\epsilon)$-approximation of $|\pi_{G-D}(u, v)|$.[14]

**Definition 3.3.** (Graph $H$)

- For a failure $f \in D$ and a tree $T \in \mathcal{T}$, if $f \in V(T)$, then we define the neighbors of $f$ in $T$ as
$$N_T(f) = \{\mathrm{parent}_T(f)\} \cup (\mathrm{children}_T(f) \cap \mathrm{Trunk}(T)).$$
In other words, $N_T(f)$ consists of the parent of $f$ in $T$, and the set of children of $f$ in $T$ which are trunk vertices. (Note that if $f \in V(T) \setminus \mathrm{Trunk}(T)$, then it is possible that $\mathrm{parent}_T(f) \notin \mathrm{Trunk}(T)$.)
- Define
$$N(f) = \bigcup_{f \in T \in \mathcal{T}} N_T(f).$$
That is, $N(f)$ is the union of $N_T(f)$'s over all trees $T \in \mathcal{T}$ such that $f \in T$.
- The vertex set of the auxiliary graph $H$ is
$$V(H) = \left( \{u, v\} \cup \bigcup_{f \in D} N(f) \right) \setminus D.$$
For each $x, y \in V(H)$, the weight of the edge $(x, y)$ in $H$ is equal to $\mathrm{DecTree}(x, y, D)$, as defined in Algorithm 2.

Note that in $V(H)$, vertices except $u$ and $v$ are defined independently from $u$ and $v$. By Lemma 2.7, for every $f \in D$ and $T \in \mathcal{T}$ containing $f$, we have $|N_T(f)| \le s + 1$. Therefore $|V(H)| \le dp \cdot 2e \ln^2 n \cdot (s+1) + 2 = O(d^{c+2} h \log^4 n)$.

**Algorithm 2** Algorithm DecTree

```
1: function DecTree(u, v, D)
2:     α ← root(u, v)
3:     while D ∩ P_α ≠ ∅ do
4:         f ← the vertex in D ∩ P_α with the highest level, breaking ties arbitrarily
5:         α ← ch(α, seg(f, P_α), l(f))      ▷ Recall l(f) is the level of f, i.e. the largest l s.t. f ∈ U_l.
6:     return |P_α|
```

[14] The vertex set of $H$ corresponds to the set of "intermediate vertices" (also called $H$) in Section 1.2.

Consider Algorithm 2. After each iteration, the set $D \cap \text{avoid}(\alpha)$ will contain at least one new vertex (namely $f$). When $|D \cap \text{avoid}(\alpha)| = d$, we will have that $D \cap P_\alpha = \emptyset$, and the algorithm terminates. Thus the algorithm executes at most $d$ iterations. It is easy to see that each iteration only requires $O(d)$ time, thus the time complexity of Algorithm 2 is $O(d^2)$.

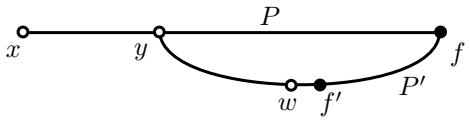
It takes $O(|V(H)|^2 d^2)$ time to build the graph $H$, and $O(|V(H)|^2)$ time to compute $|\pi_H(u, v)|$. Therefore, the query algorithm runs in $O(|V(H)|^2 d^2) = O(d^{2c+6} h^2 \log^8 n)$ time.

Since finally $D \cap P_\alpha = \emptyset$, we have the following observation:

**Observation 3.4.** *For all $u, v \in V(H)$,* $\text{DecTree}(u, v, D) \geq |\pi_{G-D}(u, v)|$.

For every $\alpha$ and $f$ considered in Algorithm 2, let $\alpha_{\text{next}} = ch(\alpha, \text{seg}(f, P_\alpha), l(f))$ be the next decision tree node the algorithm considers. If the optimal path $\pi_{G-D}(u, v)$ never intersects the set $\text{avoid}(\alpha_{\text{next}}) \setminus \text{avoid}(\alpha)$ in any iteration, then $\text{DecTree}(u, v, D)$ would indeed return the optimal path $\pi_{G-D}(u, v)$. However, if $\pi_{G-D}(u, v)$ goes through some non-failure vertex $x \in \text{avoid}(\alpha_{\text{next}}) \setminus \text{avoid}(\alpha)$, then $x$ is close to $f$, and we will show that there is some $w \in V(H)$ close to $x$, so the optimal path can be approximated by $\pi_{G-D}(u, w) \circ \pi_{G-D}(w, v)$. This is illustrated in the following important lemma. Notice that $\text{avoid}(\alpha_{\text{next}}) \setminus \text{avoid}(\alpha) = \text{seg}(f, P_\alpha) \cap U_{l(f)}$, so in this case, the level of $x$ is no less than the level of $f$, i.e. $l(x) \geq l(f)$.

**Lemma 3.5.** *Given a failed vertex $f$ and a non-failure vertex $x$ such that $l(x) \geq l(f)$, if there is a path $P$ in $G$ between $x$ and $f$ which contains no other failed vertices, then there is a vertex $w \in V(H)$ such that $|\pi_{G-D}(x, w)| \leq (2k-1)|P|$.*




*Proof.* Let $y$ be the vertex with the highest level on $P[x, f)$, and suppose $j = l(y)$. Then $l(y) = j \geq l(x) \geq l(f)$. Since there are no vertices on $P$ with level $> j$, $P$ is in $G - U_{j+1}$. Let $T_j(y, f)$ be the tree in the tree cover $\mathcal{T}_{U_{j+1}}(U_j)$, such that the distance between $y$ and $f$ in $T_j(y, f)$ is at most $(2k-1)|\pi_{G-U_{j+1}}(y, f)|$. Let $P' = \mathcal{P}_j(y, f)$ be the path between $y$ and $f$ in $T_j(y, f)$. Let $f'$ be the first failed vertex on $P'[y, f]$, and consider the predecessor $w$ of $f'$ on the path $P'[y, f]$. (That is, $P'[y, w]$ is intact from failures.) Since $P'$ is a path on the tree $T_j(y, f)$, $w$ is either the parent of $f'$ or a child of $f'$ in this tree.

- If $w$ is the parent of $f'$, then $w \in V(H)$ by the definition of $V(H)$.
- If $w$ is a child of $f'$, then $y$ is a descendant of $w$ and $f'$. Since $l(y) = j$, $y$ is a trunk vertex in $T_j(y, f)$. It follows that $w$, as an ancestor of $y$, is also a trunk vertex in $T_j(y, f)$, therefore $w \in V(H)$.

Therefore, in either case, we have $w \in V(H)$. Since

$$\begin{aligned}
|\pi_{G-D}(x, w)| &\leq |P[x, y]| + |P'[y, w]| \\
&\leq |P[x, y]| + |P'| \\
&\leq |P[x, y]| + (2k-1)|\pi_{G-U_{j+1}}(y, f)| \\
&\leq |P[x, y]| + (2k-1)|P[y, f]| \\
&\leq (2k-1)|P|,
\end{aligned}$$

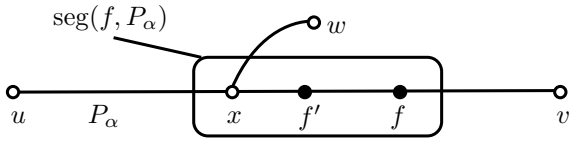
the lemma is true. □

### 3.3 Proof of Correctness

In this section, we show that $|\pi_H(u,v)| \le (1+\epsilon)|\pi_{G-D}(u,v)|$, proving the correctness of the query algorithm.

From the algorithm DecTree$(u,v,D)$, the path we get is the shortest path between $u$ and $v$ in the graph $G - \text{avoid}(\alpha_{\text{last}})$, where $\alpha_{\text{last}}$ is the last visited decision tree node of the algorithm. As we discussed before, if the real shortest path $\pi_{G-D}(u,v)$ does not go through any vertex in avoid$(\alpha_{\text{last}})$, then DecTree$(u,v,D)$ will return the correct answer. Otherwise, as avoid$(\alpha_{\text{last}})$ is the union of $\le d$ sets of the form $\text{seg}(f,P_\alpha)\cap U_i$, $\pi_{G-D}(u,v)$ must go through some vertex $x$ in a set $\text{seg}(f,P_\alpha)\cap U_i$. We can show that such $x$ will be "close" to a vertex $w$ in $V(H)$ (by Lemma 3.5), so we can use the vertices in $V(H)$ as intermediate vertices to obtain an approximate shortest path.

**Lemma 3.6.** *In the query algorithm* DecTree$(u,v,D)$*, let $\alpha$ be a decision tree node it encounters, $f \in D$ be the failed vertex which is selected in Line 4 of Algorithm 2 and $i = l(f)$. (That is, $f$ is the vertex in $D \cap P_\alpha$ with the highest level.) For any non-failure vertex $x$ in* $\text{seg}(f,P_\alpha)\cap U_i$*, there is a vertex $w \in V(H)$ such that $|\pi_{G-D}(x,w)| \le \epsilon_1 \min\{|P_\alpha[u,x]|, |P_\alpha[x,v]|\}$.*

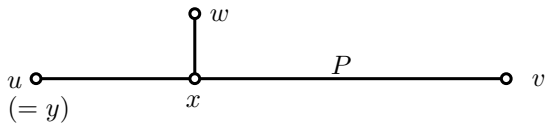



*Proof.* Let $f' \in D$ be the failed vertex closest to $x$ on the segment $\text{seg}(f,P_\alpha)$, then there are no failed vertices in $P_\alpha[x,f')$ or $P_\alpha(f',x]$. (W.l.o.g. we assume it is $P_\alpha[x,f')$.) We have $l(f) \ge l(f')$ by Algorithm 2. As $x \in U_i = U_{l(f)}$, we have $l(x) \ge l(f)$, hence $l(x) \ge l(f')$. By Lemma 3.5, there is a vertex $w \in V(H)$ such that $|\pi_{G-D}(x,w)| \le (2k-1)|P_\alpha[x,f']|$. Since $|P_\alpha[x,f']| \le \epsilon_2 \min\{|P_\alpha[u,x]|, |P_\alpha[x,v]|\}$ and $\epsilon_2 = \epsilon_1/(2k-1)$, the lemma holds. □

We show that for $u,v \in V(H)$, if the optimal path $\pi_{G-D}(u,v)$ is *not* found by DecTree$(u,v,D)$, then we can indeed find some $w \in V(H)$ such that $\pi_{G-D}(u,w)\circ\pi_{G-D}(w,v)$ is a good approximation of $\pi_{G-D}(u,v)$. Moreover, one of $\pi_{G-D}(u,w)$ or $\pi_{G-D}(w,v)$ can be dealt with by Algorithm 2, therefore we only need to "recurse" on the other one.

**Lemma 3.7.** *Let $u,v \in V(H)$ and $P = \pi_{G-D}(u,v)$. If* DecTree$(u,v,D) > |P|$*, there exist $x \in P \setminus \{u,v\}, y \in \{u,v\}, w \in V(H)$ such that*

*(a)* $|\pi_{G-D}(x,y)| \le \frac{1}{2}|\pi_{G-D}(u,v)|$,
*(b)* $|\pi_{G-D}(x,w)| \le \epsilon_1|\pi_{G-D}(x,y)|$*, and*
*(c)* DecTree$(y,w,D) \le |\pi_{G-D}(y,x)| + |\pi_{G-D}(x,w)|$*, which is smaller than* $|\pi_{G-D}(u,v)|$.




*Proof.* First we prove that there exists a triple $(x,y,w)$ that satisfies (a) and (b).

Let $\alpha$ be the last decision tree node visited by DecTree$(u,v,D)$ such that avoid$(\alpha)\cap P = \emptyset$. Since DecTree$(u,v,D) > |P|$, the procedure DecTree$(u,v,D)$ did not terminate at $\alpha$, i.e. it visited a child $\alpha_{\text{next}}$ of $\alpha$ such that $P \cap \text{avoid}(\alpha_{\text{next}}) \ne \emptyset$. Recall that $\text{avoid}(\alpha_{\text{next}}) \setminus \text{avoid}(\alpha) = \text{seg}(f,P_\alpha)\cap U_{l(f)}$

where $f$ is the failure selected by Line 4 of Algorithm 2. Therefore $P$ reaches some vertex $x \in \text{seg}(f, P_\alpha) \cap U_{l(f)}$. Since $P_\alpha$ is the shortest $u$-$v$ path in $G - \text{avoid}(\alpha)$ and $P$ is *some* $u$-$v$ path in $G - \text{avoid}(\alpha)$, we know that $|P_\alpha[u,x]| \le |P[u,x]|$ and $|P_\alpha[x,v]| \le |P[x,v]|$.

By Lemma 3.6, there is a vertex $w \in V(H)$ such that

$$|\pi_{G-D}(x,w)| \le \epsilon_1 \min\{|P_\alpha[u,x]|, |P_\alpha[x,v]|\} \le \epsilon_1 \min\{|P[u,x]|, |P[x,v]|\}.$$

Let $y$ be the endpoint in $\{u,v\}$ that is closer to $x$, then $(x,y,w)$ satisfies (a) and (b).

Among all triples $x \in P \setminus \{u,v\}, y \in \{u,v\}, w \in V(H)$ satisfying (b), we pick a triple minimizing $|\pi_{G-D}(x,y)|$, and in case of a tie choose a triple minimizing $|\pi_{G-D}(x,w)|$. It is easy to see that (a) is also satisfied. In the following we prove that (c) is satisfied.

We compare the path $P' = \pi_{G-D}(y,x) \circ \pi_{G-D}(x,w)$ between $y$ and $w$, with the path returned by DecTree$(y,w,D)$. For the sake of contradiction, suppose $|P'| <$ DecTree$(y,w,D)$. Let $\alpha'$ be the last decision tree node visited in DecTree$(y,w,D)$ such that $\text{avoid}(\alpha') \cap P' = \emptyset$. We can also see that $P'$ reaches some vertex $x' \in \text{seg}(f', P_{\alpha'}) \cap U_{l(f')}$, where $f'$ is the failure selected by Line 4 of Algorithm 2. We use Lemma 3.6 again and conclude that there is a vertex $w' \in V(H)$ such that

$$|\pi_{G-D}(x',w')| \le \epsilon_1 \min\{|P_{\alpha'}[y,x']|, |P_{\alpha'}[x',w]|\} \le \epsilon_1 \min\{|P'[y,x']|, |P'[x',w]|\}.$$

Since $x' \in P' = P'[y,x] \circ P'[x,w]$, there are two cases. (See Fig. 3.)

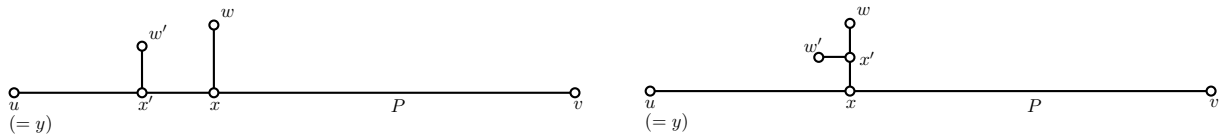



Figure 3: Two cases in the proof of Lemma 3.7.

- If $x' \in P'[y,x)$, then $|\pi_{G-D}(x',w')| \le \epsilon_1 |\pi_{G-D}(x',y)|$, i.e. the triple $(x',y,w')$ also satisfies (b). Since $|\pi_{G-D}(x',y)| < |\pi_{G-D}(x,y)|$, this contradicts our choice of $(x,y,w)$.
- If $x' \in P'[x,w]$, then $|\pi_{G-D}(x,w')| \le |P'[x,x']| + |\pi_{G-D}(x',w')| \le |P'[x,x']| + \epsilon_1 |P'[x',w]|$. As $\epsilon_1 < 1$, we have $|\pi_{G-D}(x,w')| < |\pi_{G-D}(x,w)|$, and $(x,y,w')$ also satisfies (b), contradicting our choice of $(x,y,w)$.

Hence it must be true that $|P'| \ge$ DecTree$(y,w,D)$. □

By these lemmas, we can now prove our desired approximation ratio.

**Theorem 3.8.** *For every pair $u,v \in V(H)$, the query algorithm in Section 3.2 returns an $(1+\epsilon)$-approximation of $|\pi_{G-D}(u,v)|$.*

*Proof.* It is easy to see that $|\pi_H(u,v)| \ge |\pi_{G-D}(u,v)|$ for every $u,v \in V(H)$. We prove $|\pi_H(u,v)| \le (1+\epsilon)|\pi_{G-D}(u,v)|$ below.

We sort all pairs of vertices $u,v \in V(H)$ $(u \ne v)$ by increasing order of $|\pi_{G-D}(u,v)|$, and prove by induction that $|\pi_H(u,v)| \le (1+\epsilon)|\pi_{G-D}(u,v)|$ on this order. For the $u,v$ having the smallest $|\pi_{G-D}(u,v)|$, if DecTree$(u,v,D) > |\pi_{G-D}(u,v)|$, from Lemma 3.7, there exist $y,w \in V(H)$ so that $|\pi_{G-D}(y,w)| < |\pi_{G-D}(u,v)|$, which is a contradiction. Therefore $|\pi_H(u,v)| =$ DecTree$(u,v,D) = |\pi_{G-D}(u,v)|$.

Fix some $u,v \in V(H)$, assume that for all pairs $u',v' \in V(H)$ such that $|\pi_{G-D}(u',v')| < |\pi_{G-D}(u,v)|$, it is true that $|\pi_H(u',v')| \le (1+\epsilon)|\pi_{G-D}(u',v')|$. If DecTree$(u,v,D) = |\pi_{G-D}(u,v)|$

then $|\pi_H(u,v)| \le (1+\epsilon)|\pi_{G-D}(u,v)|$ follows trivially. Otherwise we use Lemma 3.7 to obtain a triple $(x,y,w)$, where $x \in \pi_{G-D}(u,v) \setminus \{u,v\}$, $y \in \{u,v\}$, and $w \in V(H)$. We assume w.l.o.g. $y = u$, then $|\pi_{G-D}(w,x)| \le \epsilon_1 |\pi_{G-D}(u,x)|$. Since $\epsilon_1 < 1$, $|\pi_{G-D}(w,v)| \le |\pi_{G-D}(w,x)| + |\pi_{G-D}(x,v)| \le \epsilon_1|\pi_{G-D}(u,x)| + |\pi_{G-D}(x,v)| < |\pi_{G-D}(u,v)|$, thus $|\pi_H(w,v)| \le (1+\epsilon)|\pi_{G-D}(w,v)|$ by induction hypothesis. We have:

$$\begin{aligned}
|\pi_H(u,v)| &\le \mathrm{DecTree}(u,w,D) + |\pi_H(w,v)| \\
&\le |\pi_{G-D}(u,x)| + |\pi_{G-D}(x,w)| + (1+\epsilon)|\pi_{G-D}(w,v)| \\
&\le |\pi_{G-D}(u,x)| + \epsilon_1|\pi_{G-D}(u,x)| + (1+\epsilon)(\epsilon_1|\pi_{G-D}(u,x)| + |\pi_{G-D}(x,v)|) \\
&\le (1+\epsilon_1+(1+\epsilon)\epsilon_1)|\pi_{G-D}(u,x)| + (1+\epsilon)|\pi_{G-D}(x,v)| \\
&\le (1+\epsilon)|\pi_{G-D}(u,v)|.
\end{aligned}$$

□

We conclude that there is a VSDO with

$$\begin{aligned}
\text{space complexity} &\quad n^{3+1/c} \cdot O(\epsilon^{-1}\log^2 n \log(nW)/\log d)^d, \\
\text{query complexity} &\quad O(d^{2c+6}\log^{10} n/\log^2 d), \\
\text{stretch} &\quad 1+\epsilon.
\end{aligned}$$

In Section 4, we will improve the space complexity to $n^{2+1/c+o(1)}$ when $d = o\Big(\frac{\log n}{\log\log n}\Big)$ and $W = \mathrm{poly}(n)$, while increasing the query time slightly.

# 4 An $n^{2+1/c+o(1)}$-Space $(1+\epsilon)$-Stretch Oracle

In this section, we discuss the modifications needed to reduce the space complexity to $n^{2+1/c+o(1)}$. Here we set $\epsilon_3 = \frac{\epsilon}{2|V(H)|}$, and $\epsilon_4 = \epsilon_3/(4k-2)$, where $V(H)$ is the same as in Section 3 (and Lemma 3.5 still holds), while $E(H)$ are recomputed in this section. We assume that $\epsilon$ is small enough, in particular that $\epsilon < 1$ and $\epsilon_4 < \sqrt{2}-1$.

## 4.1 A Structural Theorem

Similar to [25], the main idea is, instead of storing the paths $P_\alpha$ as-is in every node $\alpha$ of $FT(u,v)$, we store an implicit representation of these paths. If the representation has size $\mathrm{poly}(\log(nW), \epsilon^{-1})$ instead of $\Omega(n)$, then our data structure has space complexity $n^{2+1/c+o(1)}$.

In [25], the authors defined *$k$-decomposable paths*, which are paths that can be represented as the concatenation of at most $k+1$ shortest paths in $G$, interleaved with at most $k$ edges. They relied on the fact (Theorem 2 of [3]) that any $k$-edge-failure shortest path is a $k$-decomposable path in $G$, therefore has a succinct representation. Unfortunately, the analogue of this statement in [25] in case of vertex failures does not hold. Even if we only remove *one* vertex (i.e. $|D| = 1$), a shortest path in $G - D$ might not be a $k$-decomposable path for $k = o(n)$.[15]

In this section, we prove a structural theorem similar to the above fact used in [25]. Before we proceed, we need some definitions.

From Lemma 3.7 we can see that for any $u, v \in V(H)$, if the path $\pi_{G-D}(u,v)$ is *$\epsilon_1$-far away from $V(H)$* in the following sense, then $\mathrm{DecTree}(u,v,D)$ indeed finds the distance between $u$ and $v$ in $G - D$:

**Definition 4.1.** We say that a path $P$ from $u$ to $v$ is *$\epsilon$-far away from $V(H)$* if there are no vertices $x \in P \setminus \{u,v\}, w \in V(H)$ such that $|\pi_{G-D}(x,w)| \le \epsilon \cdot \min\{|P[u,x]|, |P[x,v]|\}$. (See Fig. 4.)

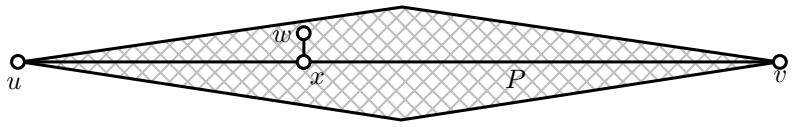


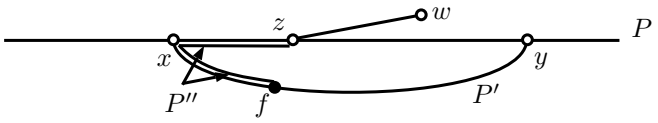

Figure 4: If $P$ is far away from $V(H)$, it means that a certain "diamond"-shaped area does not contain vertices $w \in V(H)$.

Instead of considering all $d$-failure shortest paths, we only study the ones which are $\epsilon_3$-far away from $V(H)$. We will use the concept of $k$-expath as in [25] and re-define it as *$\epsilon_4$-segment expath*. Also, instead of considering the concatenation of at most $k+1$ shortest paths in the original graph $G$, every segment here is a shortest path in some $G_i$. (Recall that $G_i$ is the induced subgraph of $G$ on all vertices of level $\leq i$.)

**Definition 4.2.** A path $P$ in $G$ is an *$\epsilon$-segment expath* if the following holds. If we partition $P$ into $\epsilon$-segments as in Definition 3.1, then for every segment $P[x,y]$, there is some $1 \leq i \leq p$ such that $P[x,y]$ is a shortest path in $G_i$.

The following structural theorem for shortest paths $\epsilon$-far away from $V(H)$ will be crucial to us. Interestingly, it is a consequence of Lemma 3.5.

**Theorem 4.3.** *For $u, v \in V(H)$, if $\pi_{G-D}(u,v)$ is $\epsilon_3$-far away from $V(H)$, then it is an $\epsilon_4$-segment expath.*




*Proof.* Let $P = \pi_{G-D}(u,v)$ and $P[x,y]$ be an $\epsilon_4$-segment of $P$. W.l.o.g. assume that $x$ and $y$ are in the first half of $P$, and $x$ is closer to $u$ than $y$. By Lemma 3.2, $|P[x,y]| \leq \epsilon_4 |P[u,x]|$. (Recall that $P[x,y]$ is an $\epsilon_4$-segment.) Consider the vertex $z$ with the highest level on $P[x,y]$, and let its level be $i = l(z)$. Then $P[x,y]$ is a path in $G_i$. If it is not the shortest path $\pi_{G_i}(x,y)$, then $\pi_{G_i}(x,y)$ must go through some failed vertex in $D$. (Since otherwise we can find a path in $G-D$ shorter than $\pi_{G-D}(u,v)$.) Let $P' = \pi_{G_i}(x,y)$ and $f$ be the failed vertex on $P'$ closest to $x$.

Since $f$ is in the graph $G_i$, we have $l(f) \leq i = l(z)$. There is a path $P'' = P[z,x] \circ P'[x,f]$ connecting $z$ and $f$ that does not go through other failed vertices. By Lemma 3.5, there is a vertex $w \in V(H)$ such that $|\pi_{G-D}(z,w)| \leq (2k-1)|P''|$. We have

$$\begin{aligned} |\pi_{G-D}(z,w)| &\leq (2k-1)(|P[z,x]| + |P'[x,f]|) \\ &\leq 2(2k-1)|P[x,y]| \\ &\leq 2(2k-1)\epsilon_4 |P[u,x]| \\ &\leq \epsilon_3 |P[u,z]|, \end{aligned}$$

which contradicts that $\pi_{G-D}(u,v)$ is $\epsilon_3$-far away from $V(H)$. Therefore, $P[x,y]$ is a shortest path in $G_i$. □

[15] Consider an unweighted graph $G = (V, E_1 \cup E_2)$ where $V = \{v_i : 0 \leq i \leq n\}$, $E_1 = \{(v_0, v_i) : 1 \leq i \leq n\}$ and $E_2 = \{(v_i, v_{i+1}) : 1 \leq i < n\}$. Then $\pi_{G-\{v_0\}}(v_1, v_n)$ is not a $0.1n$-decomposable path.

### 4.2 New Data Structure

We generalize the concept of $\epsilon_4$-segment expath to $\epsilon_4$-expath by adding more flexibility.

**Definition 4.4.** Let $B = \lceil \log_{1+\epsilon_4}(nW) \rceil$. An $\epsilon_4$-*expath* $P$ from $u$ to $v$ in $G$ is a path which is a concatenation of subpaths $P_0, \ldots, P_{2B+1}$ interleaved with at most $2B+3$ edges[16], such that the following hold.

- For every $P_k = P[u_k, v_k]$ $(0 \le k \le 2B+1)$, $P_k$ is either empty, or a shortest path in $G_i$ for some level $1 \le i \le p$.
- If $k < B+1$, then $|P[u, v_k]| \le (1+\epsilon_4)^k$; if $k \ge B+1$, then $|P[u_k, v]| \le (1+\epsilon_4)^{2B+1-k}$.

**Lemma 4.5.** *An $\epsilon_4$-segment expath $P$ from $u$ to $v$ is an $\epsilon_4$-expath.*

*Proof.* Let $j = \lfloor \log_{1+\epsilon_4}(|P|/2) \rfloor + 1$, since $1+\epsilon_4 < 2$, we have $j < B+1$. Let $P_1, \ldots, P_j$ be the $\epsilon_4$-segments (possibly empty) in the first half of $P$ such that for every $1 \le k \le j$ and $x \in P_k = P[u_k, v_k]$, $\lfloor \log_{1+\epsilon_4} |P[u, x]| \rfloor = k-1$. Then $|P[u, v_k]| \le (1+\epsilon_4)^k$, which satisfies the definition of $\epsilon_4$-expath. The second half of $P$ is symmetric. □

Recall that our data structure in Section 3 consists of $O(n^2)$ decision trees, one for each pair $u, v \in V$. Each decision tree node $\alpha$ stores a path $P_\alpha$, a subset avoid($\alpha$) of $V$, and the links to its children. The query algorithm builds an auxiliary graph $H$ on the vertex set $V(H)$ defined in Definition 3.3, and uses Algorithm 2 to determine the edge weights in $H$. At last we output $|\pi_H(u, v)|$ as the approximation of $|\pi_{G-D}(u, v)|$. Our improved data structure also fits into this high-level description, but there are some small changes:

- For every $1 \le i \le p$, we also store the shortest path distance matrix of $G_i$.
- We use $\epsilon_4$ in the definition of segments.
- In every node $\alpha \in FT(u, v)$, we store the shortest $\epsilon_4$-expath (instead of the general shortest path) from $u$ to $v$ in $G - \text{avoid}(\alpha)$, still denoted as $P_\alpha$. To save space, for every subpath $P_k = [u_k, v_k]$ which is a shortest path in some $G_i$, we only need to store a triple $(u_k, v_k, i)$.
- To check whether $f$ is in a path $P_\alpha$, for every subpath $P_k = [u_k, v_k]$ which is a shortest path in some $G_i$, we check whether $|\pi_{G_i}(u_k, f)| + |\pi_{G_i}(f, v_k)| = |\pi_{G_i}(u_k, v_k)|$. By the uniqueness assumption of shortest paths (see [33]), this method can locate a vertex in $P_\alpha$.

We now prove the correctness of this data structure, i.e. $|\pi_H(u, v)|$ is always an $(1+\epsilon)$-approximation of $|\pi_{G-D}(u, v)|$.

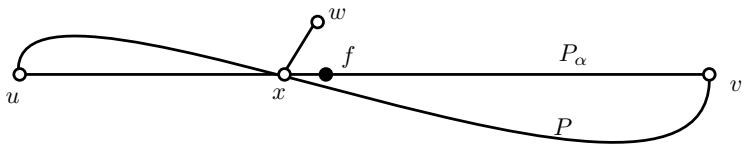
First, it is easy to check that Lemma 3.6 holds for parameter $(2k-1)\epsilon_4 = \epsilon_3/2$, as follows.

**Reminder of Lemma 3.6.** *In the query algorithm* DecTree$(u, v, D)$*, let $\alpha$ be a decision tree node it encounters, $f \in D$ be the failed vertex which is selected in Line 4 of Algorithm 2 and $i = l(f)$. (That is, $f$ is the vertex in $D \cap P_\alpha$ with the highest level.) For any non-failure vertex $x$ in* $\text{seg}(f, P_\alpha) \cap U_i$*, there is a vertex $w \in V(H)$ such that $|\pi_{G-D}(x, w)| \le (\epsilon_3/2) \min\{|P_\alpha[u, x]|, |P_\alpha[x, v]|\}$.*

Recall that Lemma 3.7 shows that, in the data structure in Section 3, any shortest path $\epsilon_1$-far away from $V(H)$ can be found by DecTree. We show that this is also true in the new data structure, where "$\epsilon_1$-far away" is changed to "$\epsilon_3$-far away".

**Lemma 4.6.** *Let $u, v \in V(H)$, and $P = \pi_{G-D}(u, v)$. If* DecTree$(u, v, D) > |P|$*, then $P$ is not $\epsilon_3$-far away from $V(H)$.*

[16] That is, the concatenation of $e_0, P_0, e_1, P_1, \ldots, P_{2B+1}, e_{2B+2}$ where each $e_i$ is either empty or an edge.



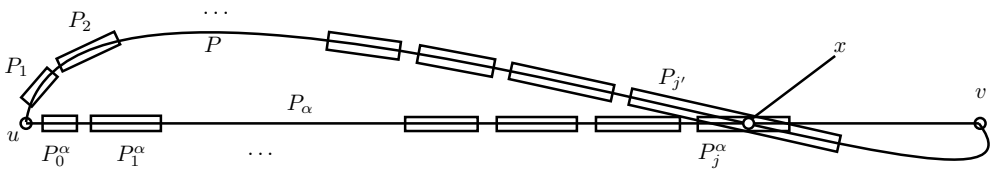

*Proof.* For the sake of contradiction, assume $P$ is $\epsilon_3$-far away from $V(H)$. By Theorem 4.3, $P$ is an $\epsilon_4$-segment expath.

Let $\alpha$ be the last decision tree node visited by $\text{DecTree}(u, v, D)$ such that $\text{avoid}(\alpha) \cap P = \emptyset$. Since $\text{DecTree}(u, v, D) > |P|$, $P$ reaches some vertex $x \in \text{avoid}(\alpha_{\text{next}}) \setminus \text{avoid}(\alpha)$, where $\alpha_{\text{next}}$ is the next decision tree node visited by $\text{DecTree}(u, v, D)$ after $\alpha$. Recall that $\text{avoid}(\alpha_{\text{next}}) \setminus \text{avoid}(\alpha) = \text{seg}(f, P_\alpha) \cap U_{l(f)}$, where $f$ is the failed vertex chosen in Line 4 of Algorithm 2. By Lemma 3.6, there is a vertex $w \in V(H)$ such that $|\pi_{G-D}(x, w)| \le (\epsilon_3/2) \min\{|P_\alpha[u, x]|, |P_\alpha[x, v]|\}$.

As $P_\alpha$ is the shortest $\epsilon_4$-expath from $u$ to $v$ in $G - \text{avoid}(\alpha)$, and $P$ is *some* such path, we have $|P| \ge |P_\alpha|$. We will prove $|P_\alpha[u, x]| \le 2|P[u, x]|$ and $|P_\alpha[x, v]| \le 2|P[x, v]|$, then it will follow that $|\pi_{G-D}(x, w)| \le \epsilon_3 \min\{|P[u, x]|, |P[x, v]\}$, contradicting that $P$ is $\epsilon_3$-far away from $V(H)$. We only prove $|P_\alpha[u, x]| \le 2|P[u, x]|$, and the case that $|P_\alpha[x, v]| \le 2|P[x, v]|$ is symmetric.

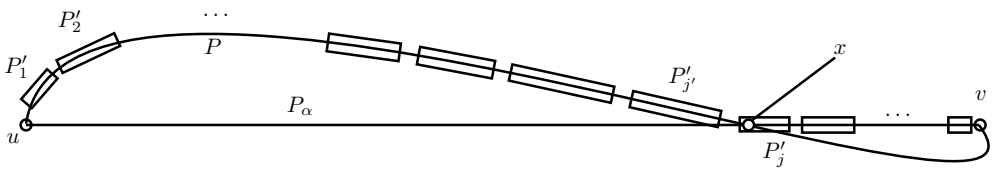
Suppose $|P[u, x]| < |P_\alpha[u, x]|/2$, we claim that the path $P[u, x] \circ P_\alpha[x, v]$ is a valid $\epsilon_4$-expath. Since $|P[u, x]| < |P_\alpha|/2 \le |P|/2$, $x$ is closer to $u$ than to $v$ in $P$. Suppose $P_\alpha$ is composed of subpaths $P^\alpha_0, \ldots, P^\alpha_{2B+1}$ interleaved with $\le 2B + 3$ edges, and $P$ is composed of segments $P_1, \ldots, P_\ell$. (Every $P^\alpha_j$ and $P_j$ is a shortest path in some $G_i$.) Recall from the proof of Lemma 4.5 that, if $x$ is in the first half of $P$, and $x \in P_k$, then $\lfloor \log_{1+\epsilon_4} |P[u, x]| \rfloor = k - 1$.

- Let $x \in P^\alpha_j$, then $j \ge \lfloor \log_{1+\epsilon_4} |P_\alpha[u, x]| \rfloor$. This is because if $j < B + 1$ (recall that $B = \lceil \log_{1+\epsilon_4}(nW) \rceil$ as in Definition 4.4), then $|P_\alpha[u, x]| \le (1 + \epsilon_4)^j$.
- Let $x \in P_{j'}$, then $j' = \lfloor \log_{1+\epsilon_4} |P[u, x]| \rfloor + 1$. Since $(1 + \epsilon_4)^2 < 2 \le |P_\alpha[u, x]|/|P[u, x]|$, we have $j' \le \lfloor \log_{1+\epsilon_4} |P_\alpha[u, x]| \rfloor - 1 \le j - 1$.




Let $P' = P[u, x] \circ P_\alpha[x, v]$, Consider the following representation of $P'$ as $P'_0, P'_1, \ldots, P'_{2B+1}$:

(i) For $0 \le i < j'$, $P'_i = P_i$.
(ii) For $i = j'$, $P'_{j'} = P_{j'}[u_{j'}, x)$, where $u_{j'}$ is the endpoint of $P_{j'}$ that lies on $P[u, x]$.
(iii) For $j' < i < j$, $P'_i = \emptyset$.
(iv) For $i = j$, $P'_j = P^\alpha_j[x, v^\alpha_j]$, where $v^\alpha_j$ is the endpoint of $P^\alpha_j$ that lies on $P_\alpha[x, v]$.
(v) For $j < i \le 2B + 1$, $P'_i = P^\alpha_i$.



We need to verify that the representation $P'_0, P'_1, \ldots, P'_{2B+1}$ satisfies the definition of $\epsilon_4$-expath. Let $u'_i, v'_i$ be the endpoints of $P'_i$, i.e. $P'_i = P'[u'_i, v'_i]$, then:

- Case I: $i \le j'$ (i.e. Items i and ii). In this case, $i < B+1$, as $P'_i$ lies in the first half of $P$. Since $|P'[u, v'_i]| = |P[u, v'_i]| \le (1+\epsilon_4)^i$, Definition 4.4 is satisfied.
- Case II: $j \le i < B+1$. In this case, $|P'[u, v'_i]| = |P[u, x]| + |P_\alpha[x, v'_i]| < |P_\alpha[u, v'_i]| \le (1+\epsilon_4)^i$, thus Definition 4.4 is satisfied.
- Case III: $j \ge B+1$. In this case, $|P'[u'_i, v]| = |P_\alpha[u'_i, v]| \le (1+\epsilon_4)^{2B+1-i}$, thus Definition 4.4 is satisfied.

We conclude that $P'$ is a valid $\epsilon_4$-expath. Since $|P'| < |P_\alpha|$, this contradicts the choice of $P_\alpha$.

Therefore $|P_\alpha[u, x]| \le 2|P[u, x]|$, and by symmetry, $|P_\alpha[x, v]| \le 2|P[x, v]|$. It follows that $P$ is not $\epsilon_3$-far away from $V(H)$. □

We prove the following theorem that immediately implies the approximation ratio of the algorithm.

**Theorem 4.7.** *For every pair $u, v \in V(H)$, $|\pi_H(u, v)| \le (1+\epsilon)|\pi_{G-D}(u, v)|$.*

*Proof.* For the purpose of the proof, we construct a subgraph $H'$ of $H$ on the same set of vertices (i.e. $V(H)$), but only keep the edges $(u, v)$ where $\pi_{G-D}(u, v)$ is $\epsilon_3$-far away from $V(H)$. By Lemma 4.6, the weight of every single edge $(u, v)$ in $H'$ is exactly $|\pi_{G-D}(u, v)|$.

We sort all pairs of vertices $u, v \in V(H)$ by nondecreasing order of $|\pi_{G-D}(u, v)|$. For every $u, v \in V(H)$, we define a $u$-$v$ path in $H'$ inductively in this order, and denote it as $p(u, v)$. The path $p(u, v)$ is defined as follows.

- If $\pi_{G-D}(u, v)$ is $\epsilon_3$-far away from $V(H)$, $p(u, v)$ consists of a single edge $(u, v)$.
- If $\pi_{G-D}(u, v)$ is not $\epsilon_3$-far away from $V(H)$, there exist $x \in \pi_{G-D}(u, v) \setminus \{u, v\}, w \in V(H)$ such that $|\pi_{G-D}(x, w)| \le \epsilon_3 \min\{|\pi_{G-D}(u, x)|, |\pi_{G-D}(x, v)|\}$. Since $\epsilon_3 < 1$, $|\pi_{G-D}(u, w)|$ and $|\pi_{G-D}(w, v)|$ are both smaller than $|\pi_{G-D}(u, v)|$, so $p(u, w)$ and $p(w, v)$ are both well-defined. We concatenate these paths to form $p(u, v)$, i.e. we define $p(u, v) = p(u, w) \circ p(w, v)$.

Let $k(u, v)$ be the number of edges in $p(u, v)$. We prove that for every $u, v \in V(H)$,

$$|\pi_{H'}(u, v)| \le (1+\epsilon_3)^{k(u,v)} |\pi_{G-D}(u, v)|.$$

We proceed by induction on $k(u, v)$. When $k(u, v) = 1$, $|\pi_{H'}(u, v)| = |\pi_{G-D}(u, v)|$. Assume this is true for all pairs $(u, v)$ such that $k(u, v) < j$, consider some $(u, v)$ such that $k(u, v) = j$. Let $x, w$ be the vertices selected in the construction of $p(u, v)$, then both $k(u, w)$ and $k(w, v)$ are less than $j$. As $|\pi_{G-D}(x, w)| \le (\epsilon_3/2)|\pi_{G-D}(u, v)|$, we have

$$\begin{aligned}
|\pi_{H'}(u, v)| &\le |\pi_{H'}(u, w)| + |\pi_{H'}(w, v)| \\
&\le (1+\epsilon_3)^{j-1}(|\pi_{G-D}(u, w)| + |\pi_{G-D}(w, v)|) \\
&\le (1+\epsilon_3)^{j-1}(|\pi_{G-D}(u, v)| + 2|\pi_{G-D}(x, w)|) \\
&\le (1+\epsilon_3)^{j-1}(|\pi_{G-D}(u, v)| + 2\frac{\epsilon_3}{2}|\pi_{G-D}(u, v)|) \\
&\le (1+\epsilon_3)^{j}|\pi_{G-D}(u, v)|.
\end{aligned}$$

Thus, for every $u, v \in V(H)$,

$$\begin{aligned}
|\pi_H(u,v)| &\le |\pi_{H'}(u,v)| \\
&\le \left(1 + \frac{\epsilon}{2|V(H)|}\right)^{|V(H)|} |\pi_{G-D}(u,v)| \\
&\le e^{\frac{\epsilon}{2}} |\pi_{G-D}(u,v)| \\
&< (1+\epsilon)|\pi_{G-D}(u,v)|. && \text{(since } \epsilon < 1\text{)}
\end{aligned}$$

□

Each $\epsilon_4$-expath can be stored in $O(\epsilon_4^{-1}\log(nW))$ space. Each non-leaf node in the decision tree has $O(h\epsilon_4^{-1}\log(nW))$ children. Thus we have a VSDO of

$$\begin{aligned}
\text{space complexity} &\quad n^{2+1/c}\epsilon_4^{-1}\log(nW)\cdot O(h\epsilon_4^{-1}\log(nW))^d, \\
\text{query complexity} &\quad O(d^2|V(H)|^2\cdot\epsilon_4^{-1}\log(nW)), \\
\text{stretch} &\quad 1+\epsilon.
\end{aligned}$$

As $\epsilon_4^{-1} = O(|V(H)|\cdot\epsilon^{-1}\log n) = O(d^{c+2}\epsilon^{-1}h\log^5 n)$, the VSDO is of

$$\begin{aligned}
\text{space complexity} &\quad n^{2+1/c}\cdot(\epsilon^{-1}d^c\log(nW))^{O(d)}, \\
\text{query complexity} &\quad \tilde{O}(\epsilon^{-1}d^{3c+8}\log W), \\
\text{stretch} &\quad 1+\epsilon.
\end{aligned}$$

We improve both the space complexity and query time in the next subsection.

### 4.3 An Improvement

In Section 4.2, we use $\epsilon_4$-segments in the decision tree. Therefore, each decision tree node that is not a leaf has $O(\epsilon_4^{-1}\cdot h\log(nW))$ children, and each decision tree node occupies $O(\epsilon_4^{-1}\cdot\log(nW))$ space. As $\epsilon_4^{-1} = \Theta(|V(H)|\cdot\epsilon^{-1}\log n)$, this $\epsilon_4^{-1}$ factor may seem too large. In this section, we show that the $|V(H)|$ factor in $\epsilon_4^{-1}$ can be shaved.

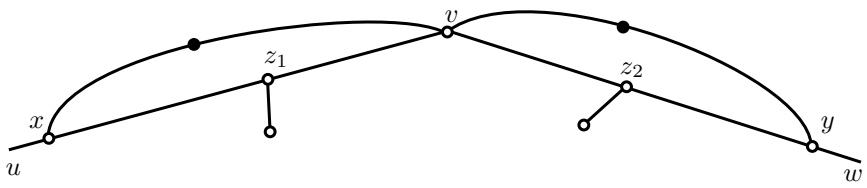
Let $\epsilon_1 = \epsilon/(2+\epsilon)$ as in Section 3 and $\epsilon_5 = \epsilon_1/(4k-2)$. We will use $O(\epsilon_5^{-1}\log(nW))$ space to represent a node in the decision tree $FT(u,v)$. A first attempt would be to store the shortest $\epsilon_5$-expath in each node $\alpha$, but we face a technical problem as follows. Suppose DecTree$(u,v,D)$ does *not* capture the shortest path $P = \pi_{G-D}(u,v)$, then by Lemma 3.7, $P$ is *not* far from $V(H)$. In other words, there are vertices $x \in P$ and $w \in V(H)$ such that $\pi_{G-D}(x,w) \le \epsilon_1|P[u,x]|$. (Here we assume w.l.o.g. that $x$ is closer to $u$.) Let $P_1 = \pi_{G-D}(u,x)\circ\pi_{G-D}(x,w)$, and $P_2 = \pi_{G-D}(w,v)$, we "recursively" find $P_1$ and $P_2$ and concatenate them as an approximation of $P$. The proof of Lemma 3.7 shows that $P_1$ is far away from $V(H)$, so we may attempt to use Lemma 4.6 to conclude that $|\text{DecTree}(u,w,D)| \le |P_1|$, and we only need to "recurse" on $P_2$. However, Lemma 4.6 relies on Theorem 4.3, which requires $P_1$ to be a *shortest* path in $G-D$, while $P_1 = \pi_{G-D}(u,x)\circ\pi_{G-D}(x,w)$ is not necessarily the shortest $u$-$w$ path.

The solution is simple. If $P_1$ is $\epsilon_1$-far from $V(H)$, we can use the same proof method of Theorem 4.3, to prove that each segment of $P_1 = \pi_{G-D}(u,x)\circ\pi_{G-D}(x,w)$ is *the concatenation of at most two* shortest paths in some $G_i$ and $G_j$. (The original Theorem 4.3 proved that each segment of $\pi_{G-D}(u,v)$ is a shortest path in some $G_i$.) Therefore, we define *segment bipaths*, in which each segment is the concatenation of two shortest paths in $G_i$ and $G_j$, rather than one shortest path in $G_i$ as in segment expaths.

**Definition 4.8.** A path $P$ in $G$ is an $\epsilon_5$-*segment bipath* if the following holds. If we partition $P$ into $\epsilon_5$-segments as in Definition 3.1, for every segment $P[u_k,v_k]$, there exist two levels $i,j$ and a vertex $z \in P[u_k,v_k]$ such that $P[u_k,v_k] = \pi_{G_i}(u_k,z)\circ\pi_{G_j}(z,v_k)$.

The following theorem can be proved by similar arguments as Theorem 4.3.

**Theorem 4.9.** *For $u, v, w \in V$, let $P = \pi_{G-D}(u, v) \circ \pi_{G-D}(v, w)$. If $P$ is $\epsilon_1$-far away from $V(H)$, then it is an $\epsilon_5$-segment bipath.*




*Proof Sketch.* Let $P[x, y]$ be a segment of $P$. If $v \notin P[x, y]$ then the argument of Theorem 4.3 applies to $P[x, y]$, and there is some $1 \le i \le p$ such that $P[x, y] = \pi_{G_i}(x, y)$. If $v \in P[x, y]$, then $P[x, v]$ and $P[v, y]$ are shortest paths in $G - D$ respectively. Let $z_1$ be the vertex with the highest level in $P[x, v]$, and $z_2$ be the vertex with the highest level in $P[v, y]$. We proceed with the same argument as in Theorem 4.3, and we can see that $P[x, v]$ is the shortest $x$-$v$ path in $G_{l(z_1)}$, and $P[v, y]$ is the shortest $v$-$y$ path in $G_{l(z_2)}$. □

Similarly we can define $\epsilon_5$-*bipaths*:

**Definition 4.10.** Let $B = \lceil \log_{1+\epsilon_5}(nW) \rceil$. An $\epsilon_5$-*bipath* $P$ from $u$ to $v$ in $G$ is a path which is a concatenation of subpaths $P_0, \ldots, P_{2B+1}$ interleaved with at most $2B + 3$ edges, such that the following hold.

- For every $P_k = P[u_k, v_k]$ $(0 \le k \le 2B + 1)$, either $P_k$ is empty, or there exists a vertex $z \in P[u_k, v_k]$ and two levels $1 \le i, j \le p$, such that $P[u_k, v_k] = \pi_{G_i}(u_k, z) \circ \pi_{G_j}(z, v_k)$.
- If $k < B + 1$, then $|P[u, v_k]| \le (1 + \epsilon_5)^k$; if $k \ge B + 1$, then $|P[u_k, v]| \le (1 + \epsilon_5)^{2B+1-k}$.

We also use $\epsilon_5$ in the definition of segments when constructing decision trees $FT(u, v)$. In each node $\alpha \in FT(u, v)$, we store the shortest $\epsilon_5$-bipath from $u$ to $v$ as the path $P_\alpha$. Lemma 3.6 still holds (for parameter $(2k - 1)\epsilon_5 = \epsilon_1/2$).

**Reminder of Lemma 3.6.** *In the query algorithm* DecTree$(u, v, D)$*, let $\alpha$ be a decision tree node it encounters, $f \in D$ be the failed vertex which is selected in Line 4 of Algorithm 2 and $i = l(f)$. (That is, $f$ is the vertex in $D \cap P_\alpha$ with the highest level.) For any non-failure vertex $x$ in* $\mathrm{seg}(f, P_\alpha) \cap U_i$*, there is a vertex $w \in V(H)$ such that $|\pi_{G-D}(x, w)| \le (\epsilon_1/2) \min\{|P_\alpha[u, x]|, |P_\alpha[x, v]|\}$.*

It is easy to verify that the counterparts of Lemma 4.5 and Lemma 4.6 also hold for (segment) bipaths.

**Lemma 4.11.** *An $\epsilon_5$-segment bipath $P$ from $u$ to $v$ is an $\epsilon_5$-bipath.*

**Lemma 4.12.** *(Assume $1 + \epsilon_5 < \sqrt{2}$.) Let $y, w \in V(H)$, $x \in V$, and $P = \pi_{G-D}(y, x) \circ \pi_{G-D}(x, w)$. If* DecTree$(y, w, D) > |P|$*, then $P$ is not $\epsilon_1$-far away from $V(H)$.*

*Proof Sketch of Lemma 4.11 and Lemma 4.12.* The arguments are essentially the same as Lemmas 4.5 and 4.6, except that each subpath in $P$ and $P_\alpha$ is now a concatenation of two shortest paths in $G_i$ and $G_{i'}$. This does not affect the calculation of lengths of paths in the proofs. In particular, in Lemma 4.6, the representation of $P[u, x] \circ P_\alpha[x, v]$ as $\epsilon_5$-bipath remains exactly the same, and it is easy to verify the validity of $P[u, x] \circ P_\alpha[x, v]$ as an $\epsilon_5$-bipath. □

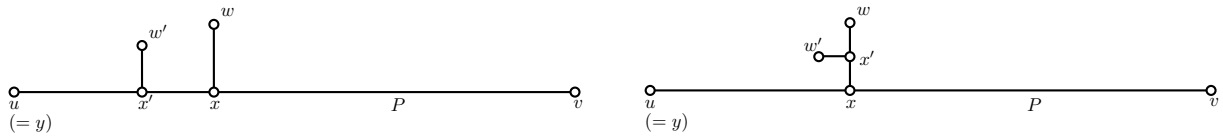
Recall that the query algorithm builds the graph $H$ on vertex set $V(H)$, adds an edge of weight $\text{DecTree}(x, y, D)$ for each $x, y \in V(H)$, and outputs the value $|\pi_H(u, v)|$. We now prove that the query algorithm has stretch $1 + \epsilon$.

**Theorem 4.13.** *For every $u, v \in V(H)$, $|\pi_H(u, v)| \le (1 + \epsilon)|\pi_{G-D}(u, v)|$.*

*Proof.* For all pairs $u, v \in V(H)$, we sort the lengths $|\pi_{G-D}(u, v)|$ in nondecreasing order, and use induction on this order. For each $u, v \in V(H)$, if $\pi_{G-D}(u, v)$ is $\epsilon_1$-far away from $V(H)$, by Lemma 4.12, $\text{DecTree}(u, v, D) = |\pi_{G-D}(u, v)|$ and we are done. Otherwise let $P = \pi_{G-D}(u, v)$, then there are vertices $x \in P \setminus \{u, v\}$, $y \in \{u, v\}$ and $w \in V(H)$ such that $|P[x, y]| \le \frac{1}{2}|P|$ and $|\pi_{G-D}(x, w)| \le \epsilon_1 |P[x, y]|$.

Among all such triples $(x, y, w)$, we choose the triple that minimizes $|P[x, y]|$, and in case of ties choose the triple that minimizes $|\pi_{G-D}(x, w)|$. W.l.o.g. assume $y = u$. Let $P' = \pi_{G-D}(u, x) \circ \pi_{G-D}(x, w)$, if $P'$ is not $\epsilon_1$-far away from $V(H)$, then there are vertices $x' \in P' \setminus \{u, w\}, y' \in \{u, w\}$ and $w' \in V(H)$ such that $|\pi_{G-D}(x', w')| \le \epsilon_1 |P'[x', y']|$. The same argument as Lemma 3.7 shows that this is a contradiction to the choice of $(x, y, w)$:

- If $x' \in P'[u, x)$, then the triple $(x', y, w')$ also satisfies that $|\pi_{G-D}(x', w')| \le \epsilon_1 |P[x', y]|$, and $|P[x', y]| < |P[x, y]|$. So we should have chosen the triple $(x', y, w')$ instead of $(x, y, w)$.
- If $x' \in P'[x, w]$, then $|\pi_{G-D}(x, w')| \le |P'[x, x']| + |\pi_{G-D}(x', w')| \le |P'[x, x']| + \epsilon_1 |P'[x', w]| < |\pi_{G-D}(x, w)|$ as $\epsilon_1 < 1$. So we should have chosen the triple $(x, y, w')$ instead of $(x, y, w)$.




It follows that $P'$ is $\epsilon_1$-far away from $V(H)$. By Lemma 4.12, we have $\text{DecTree}(u, w, D) \le |P'| = |\pi_{G-D}(u, x)| + |\pi_{G-D}(x, w)|$. It is easy to see that $|\pi_{G-D}(w, v)| < |\pi_{G-D}(u, v)|$, thus by induction hypothesis $|\pi_H(w, v)| \le (1 + \epsilon)|\pi_{G-D}(w, v)|$. We have

$$\begin{aligned}
|\pi_H(u, v)| &\le |\pi_H(u, w)| + |\pi_H(w, v)| \\
&\le |\pi_{G-D}(u, x)| + |\pi_{G-D}(x, w)| + (1 + \epsilon)(|\pi_{G-D}(w, x)| + |\pi_{G-D}(x, v)|) \\
&\le |\pi_{G-D}(u, x)|(1 + \epsilon_1 + (1 + \epsilon)\epsilon_1) + (1 + \epsilon)|\pi_{G-D}(x, v)| \\
&= (1 + \epsilon)|\pi_{G-D}(u, v)|.
\end{aligned}$$

□

Since an $\epsilon_5$-bipath occupies $O(\epsilon_5^{-1} \log(nW))$ space, and each non-leaf node has $O(h\epsilon_5^{-1} \log(nW))$ children, we have a VSDO of

$$\begin{aligned}
\text{space complexity} &\quad n^{2+1/c}\epsilon_5^{-1} \log(nW) \cdot O(h\epsilon_5^{-1} \log(nW))^d, \\
\text{query complexity} &\quad O(d^2 |V(H)|^2 \cdot \epsilon_5^{-1} \log(nW)), \\
\text{stretch} &\quad 1 + \epsilon.
\end{aligned}$$

As $\epsilon_5^{-1} = O(\epsilon^{-1} \log n)$, the VSDO is of

$$\begin{aligned}
\text{space complexity} &\quad n^{2+1/c} \cdot (\log d / \log n) \cdot O(\epsilon^{-1} \log^2 n \log(nW) / \log d)^{d+1}, \\
\text{query complexity} &\quad O(\epsilon^{-1} d^{2c+6} \log^{11} n \log(nW) / \log^2 d), \\
\text{stretch} &\quad 1 + \epsilon.
\end{aligned}$$

### 4.4 Implementation Details

**Preprocessing.** Given a subgraph $G'$ of $G$, vertices $s, t \in V$ and $\epsilon' > 0$, we show that the shortest $\epsilon'$-expath from $s$ to $t$ in $G'$ can be computed in polynomial time.

Let $\pi'_{G'}(s,t)$ be the shortest path of the form $\pi_{G_i}(s,t)$, where $1 \le i \le p$ and $\pi_{G_i}(s,t) \subseteq G'$. (Note that $\pi'_{G'}(s,t)$ may not exist). First we compute $\pi'_{G'}(s,t)$ for all pairs of $s, t \in V$. Then let $\pi(s,t,j)$ be the shortest $s$-$t$ path $P$ in $G'$ such that the following hold.

- $P$ is the concatenation of subpaths $P_0, \ldots, P_j$ interleaved with $\le j+1$ edges. Moreover, denote $P_k = P[u_k, v_k]$, where $u_k, v_k$ are endpoints of $P_k$ and $u_k$ is the one closer to $u$, then $v_j = t$, but there might be an edge between $s$ and $u_0$. (That is, $P$ is the concatenation of $e_0, P_0, e_1, P_1, \ldots, e_j, P_j$ where each $e_i$ is an edge and each $P_i$ is a subpath.)
- For every $0 \le k \le j$, $P_k$ is either empty or a shortest path in $G_i$ for some level $1 \le i \le p$.
- For every $0 \le k \le j$, $|P[s, v_k]| \le (1+\epsilon')^k$.

We use a dynamic programming algorithm to compute $|\pi(s,t,k)|$ for all $k \le B$. To start with, we artificially define $|\pi(s,t,-1)|$ as:

$$|\pi(s,t,-1)| = \begin{cases} 0 & \text{if } s = t \\ +\infty & \text{if } s \neq t \end{cases}.$$

Given $\{|\pi(s,t,j-1)|\}$ for all $s, t \in V$, we compute $|\pi(s,t,j)|$ as follows:

$$|\pi(s,t,j)| = \begin{cases} |\tilde{\pi}(s,t,j)| & \text{if } |\tilde{\pi}(s,t,j)| \le (1+\epsilon')^j \\ +\infty & \text{otherwise} \end{cases},$$

where

$$|\tilde{\pi}(s,t,j)| = \min_{(u,v)} \{|\pi(s,u,j-1)| + w(u,v) + |\pi'_{G'}(v,t)|\}. \tag{3}$$

Then the length of shortest $\epsilon'$-expath is

$$\min_{(u,v)} \{|\pi(s,u,B)| + w(u,v) + |\pi(v,t,B)|\}.$$

Here $w(u,v)$ is the weight of the edge between $u$ and $v$. If $u = v$ then we assume $w(u,v) = 0$.

We can easily adapt the algorithm to obtain the actual shortest $\epsilon'$-expath.

If we replace the term $\pi'_{G'}(s,t)$ in (3) by $\pi''_{G'}(s,t)$, which is defined as the shortest concatenated path of the form $\pi'_{G'}(s,u) \circ \pi'_{G'}(u,t)$, then we can also compute shortest $\epsilon'$-bipaths in polynomial time. Once we have a polynomial-time algorithm for computing the shortest $\epsilon'$-expath or $\epsilon'$-bipath in a subgraph $G'$, it is easy to see that the whole preprocessing time is polynomial in the space complexity.

**Query.** An $\epsilon'$-expath from $u$ to $v$ is stored as $O(\epsilon'^{-1} \log(nW))$ triples $(x, y, l)$, where each triple denotes a subpath $\pi_{G_l}(x,y)$. To check whether a failed vertex $f$ is in an $\epsilon'$-expath $P_\alpha$, we check every subpath $\pi_{G_l}(x,y)$ whether it contains $f$ by checking whether $\pi_{G_l}(x,f) + \pi_{G_l}(f,y) = \pi_{G_l}(x,y)$. The correctness of this method relies on the uniqueness assumption of shortest paths. If $f$ is in $P_\alpha$, we can also find the segment it is in, by computing $\lfloor \log_{1+\epsilon'} |P_\alpha[u,f]| \rfloor$ or $\lfloor \log_{1+\epsilon'} |P_\alpha[f,v]| \rfloor$.

If we store the distance matrices of each $G_i$ during preprocessing, then every operation (i.e. checking if $f \in P_\alpha$ and locating $\mathrm{seg}(f, P_\alpha)$) can be done in $O(\epsilon'^{-1} \log(nW))$ time. Therefore the time complexity of Algorithm 2 becomes $O(\epsilon'^{-1} \log(nW) \cdot d^2)$. Similar arguments also apply to $\epsilon'$-bipaths.

**Retrieving the actual path.** The actual $(1+\epsilon)$-approximate shortest path can be efficiently retrieved as follows. (By *retrieving a path efficiently*, we mean finding it in $O(\ell)$ additional time, where $\ell$ is the number of vertices in the path.)

- For every $1 \le i \le p$, we also preprocess the shortest paths of $G_i$. That is, for every $v \in V(G_i)$, we precompute the incoming shortest path tree rooted at $v$. Consequently, given any $1 \le i \le p$ and $u, v \in V(G_i)$, we can retrieve the path $\pi_{G_i}(u, v)$ efficiently.
- Let $\alpha \in FT(u, v)$ be a decision tree node. Recall that $P_\alpha$ is an $\epsilon_4$-expath or an $\epsilon_5$-bipath, therefore a concatenation of $O(\epsilon_4^{-1} \log(nW))$ or $O(\epsilon_5^{-1} \log(nW))$ paths of the form $\pi_{G_i}(x, y)$. Hence, $P_\alpha$ can be retrieved efficiently.
- Let $u, v \in V$ and $D$ be a set of failed vertices. We build the graph $H$ according to Definition 3.3, and find the shortest $u$-$v$ path in $H$. Each edge $(x, y)$ in this path corresponds to a path returned by DecTree$(x, y, D)$, which by Algorithm 2 is $P_\alpha$ for some decision tree node $\alpha$. The concatenation of these paths $P_\alpha$ for each edge on $\pi_H(u, v)$ forms an $(1+\epsilon)$-approximate shortest $u$-$v$ path in $G - D$. As each $P_\alpha$ can be retrieved efficiently, this path can also be retrieved efficiently.

### 4.5 A Reduction from Arbitrary Weights to Bounded Weights

If $W = n^{\omega(1)}$, then we may be unsatisfied with the $\log^d(nW)$ factor in the space complexity of our oracle. We can replace the $\log^d(nW)$ factor by $\log W \log^{d-1} n$ in the space complexity of our data structure, via a reduction from arbitrary weights to bounded weights. This reduction appears in [25, Lemma 4.1] and we notice that it also holds for vertex failures.

**Lemma 4.14** (Lemma 4.1 of [25], rephrased)**.** *Suppose we have a VSDO for undirected graphs with edge weights in $[1, n^3]$, which occupies $S$ space, needs $Q$ query time and has stretch $A$. Then we can build a VSDO for undirected graphs with edge weights in $[1, W]$, which occupies $O(S \log W / \log n)$ space, needs $O(Q \log \log W)$ query time and has stretch $(1 + 1/n)A$.*

*Proof.* For every $0 \le i \le \frac{\log W}{\log n}$, we build a VSDO $\mathcal{O}^i$ on the graph $\tilde{G}^i$, which is defined as follows: $V(\tilde{G}^i) = V(G)$ and for each edge $(u, v)$ of weight $w$ in $G$, if $w \le n^{i+1}$, then we have an edge $(u, v)$ of weight $\lceil w \cdot n^{-(i-2)} \rceil$ in $\tilde{G}^i$. Note that the graphs $\tilde{G}^i$ are *monotone* in the sense that, if an edge appears in $\tilde{G}^i$, then it also appears (albeit with a different weight) in $\tilde{G}^{i+1}$. Also note that the edge weights in every $\tilde{G}^{i+1}$ is at most $n^3$.

Given a query $(u, v, D)$, we can use binary search to find the smallest integer $i$ such that $s$ and $t$ are connected in $\tilde{G}^i - D$. Then we use the oracle $\mathcal{O}^i$ and $\mathcal{O}^{i+1}$ to compute an $A$-approximation of the value

$$ans = \min\left\{\delta_{\tilde{G}^i - D}(u, v) \cdot n^{i-2}, \delta_{\tilde{G}^{i+1} - D}(u, v) \cdot n^{i-1}\right\}.$$

It remains to prove that $\delta_{G-D}(u, v) \le ans \le (1 + 1/n)\delta_{G-D}(u, v)$. That $\delta_{G-D}(u, v) \le ans$ is trivial. Let $\tilde{W}$ be the largest edge weight in $\pi_{G-D}(u, v)$, and $i_\star = \lfloor \log_n \tilde{W} \rfloor$. Since $\tilde{W} \le n^{i_\star+1}$, $u, v$ are connected in $\tilde{G}^{i_\star}$. On the other hand, every edge in $G$ that appears in $\tilde{G}^{i_\star-2}$ has weight at most $n^{i_\star-1}$, thus if $u, v$ are connected in $\tilde{G}^{i_\star-2}$, then $\delta_{G-D}(u, v) \le (n-1)n^{i_\star-1} < n^{i_\star}$, contradicting the definition of $i_\star$. Therefore $i_\star - 1 \le i \le i_\star$ and $i_\star \in \{i, i+1\}$.

We have $ans \le \delta_{\tilde{G}^{i_\star} - D}(u, v) \cdot n^{i_\star-2}$. For every edge $e \in E(G)$ with weight $w$, if $e$ appears in the graph $\tilde{G}^{i_\star}$, then $(\lceil w \cdot n^{-(i_\star-2)} \rceil \cdot n^{i_\star-2}) \le w + n^{i_\star-2}$, i.e. every such edge is "overestimated" by an additive error of at most $n^{i_\star-2}$. It follows that $ans \le \delta_{G-D}(u, v) + (n-1)n^{i_\star-2}$. Since $\delta_{G-D}(u, v) \ge n^{i_\star}$, we have $ans \le (1 + 1/n)\delta_{G-D}(u, v)$.

As our new oracle computes an $A$-approximation of $ans$, its stretch is $(1 + 1/n)A$. □

Assuming $\epsilon > 2/n$, Lemma 4.14 transforms the VSDO in Section 3 into a VSDO of

$$\begin{aligned} \text{space complexity} &\quad n^{3+1/c}(\log W/\log n)\cdot O(\epsilon^{-1}\log^3 n/\log d)^d, \\ \text{query complexity} &\quad O(d^{2c+6}\log^{10} n \log\log W/\log^2 d), \\ \text{stretch} &\quad 1+\epsilon, \end{aligned}$$

and the VSDO in Section 4 into a VSDO of

$$\begin{aligned} \text{space complexity} &\quad n^{2+1/c}(\log W\log d/\log^2 n)\cdot O(\epsilon^{-1}\log^3 n/\log d)^{d+1}, \\ \text{query complexity} &\quad O(\epsilon^{-1}d^{2c+6}\log^{12} n \log\log W/\log^2 d), \\ \text{stretch} &\quad 1+\epsilon. \end{aligned}$$

# 5 A $\mathrm{poly}(\log n, d)$-Stretch Oracle

We present an oracle of space complexity $n^{2+1/c}\mathrm{poly}(\log(nW), d)$ that achieves $\mathrm{poly}(\log n, d)$ stretch and $\mathrm{poly}(\log(nW), d)$ query time. We actually consider a decision version of our problem, namely:

a) It is given a parameter $\rho$.
b) If $\delta_{G-D}(u,v) \le \rho$, the data structure outputs YES.
c) For some $A = \mathrm{poly}(\log n, d)$, if $\delta_{G-D}(u,v) > \rho \cdot A$, then the data structure outputs NO.

A standard binary search argument shows that if the above decision version can be solved in space $S$, query time $Q$ and stretch $A$, then there is a VSDO of size $O(S\log(nW))$, query time $O(Q\log\log(nW))$ and stretch $2A$. (See also [26].) Let $\mathcal{O}_\rho$ be the oracle solving the decision version, and we build a VSDO $\mathcal{O}^\star$ as follows. The new oracle $\mathcal{O}^\star$ consists of $O(\log(nW))$ old oracles $\{\mathcal{O}_{2^i} : 0 \le i \le \lceil\log_2(nW)\rceil\}$. For convenience we assume $\mathcal{O}_{2^{-1}}$ always outputs NO and $\mathcal{O}_{2^{\lceil\log_2(nW)\rceil+1}}$ always outputs YES.

On a query $u, v, D$, the oracle $\mathcal{O}^\star$ finds some $i$ ($0 \le i \le \lceil\log_2(nW)\rceil + 1$) such that $\mathcal{O}_{2^{i-1}}$ outputs NO and $\mathcal{O}_{2^i}$ outputs YES, and outputs $ans = A \cdot 2^i$. Such $i$ always exists and can be found in $O(\log\log(nW))$ oracle calls by binary search.[17] Since $\mathcal{O}_{2^{i-1}}$ outputs NO, we have $\delta_{G-D}(u,v) \ge 2^{i-1}$ by b), so $ans \le 2A \cdot \delta_{G-D}(u,v)$. Since $\mathcal{O}_{2^i}$ outputs YES, we have $\delta_{G-D}(u,v) \le A \cdot 2^i$ by c), so $ans \ge \delta_{G-D}(u,v)$. Thus $\mathcal{O}^\star$ is indeed a VSDO with stretch $2A$.

## 5.1 Preliminaries

### 5.1.1 $k$-Covering Sets

Denote $[l,r] = \{l, l+1, \ldots, r\}$ and $[n] = [1,n]$. An *interval* is a set of the form $[l,r]$. Given a universe $[n]$, a set of intervals $\mathcal{I}$ is a *$k$-covering set* of $[n]$ if for every $1 \le l \le r \le n$, there are at most $k$ intervals $I_1, I_2, \ldots, I_k \in \mathcal{I}$ such that $\bigcup_{i=1}^k I_i = [l,r]$.

The notion of $k$-covering sets arise from the study of the *semigroup range query* problem [5, 72], which is a generalization of the *range minimum query* problem [4, 10, 43]. For example, by constructing an interval tree over $[n]$, it is easy to see that there is an $O(\log n)$-covering set of $[n]$ whose size is $O(n)$. We use the following (stronger) results of [5, 72]:

**Theorem 5.1.** *There exists a polynomial-time computable $O(\alpha(n))$-covering set $\mathcal{I}$ of $[n]$ with $|\mathcal{I}| = O(n)$, where $\alpha(n)$ is the inverse-Ackermann function. Moreover, for any interval $[l,r]$, we can find $O(\alpha(n))$ intervals whose union is $[l,r]$ in $O(\alpha(n))$ time.*

[17] We maintain an interval $[l,r]$ such that on this query, $\mathcal{O}_{2^l}$ outputs NO and $\mathcal{O}_{2^r}$ outputs YES. Initially $l = -1$ and $r = \lceil\log_2(nW)\rceil + 1$. In each iteration, let $m = \lfloor(l+r)/2\rfloor$, and we query $\mathcal{O}_{2^m}$. If $\mathcal{O}_{2^m}$ returns NO, we set $l \leftarrow m$, otherwise we set $r \leftarrow m$. After $O(\log\log(nW))$ oracle calls, we have $r - l = 1$ and we are done.

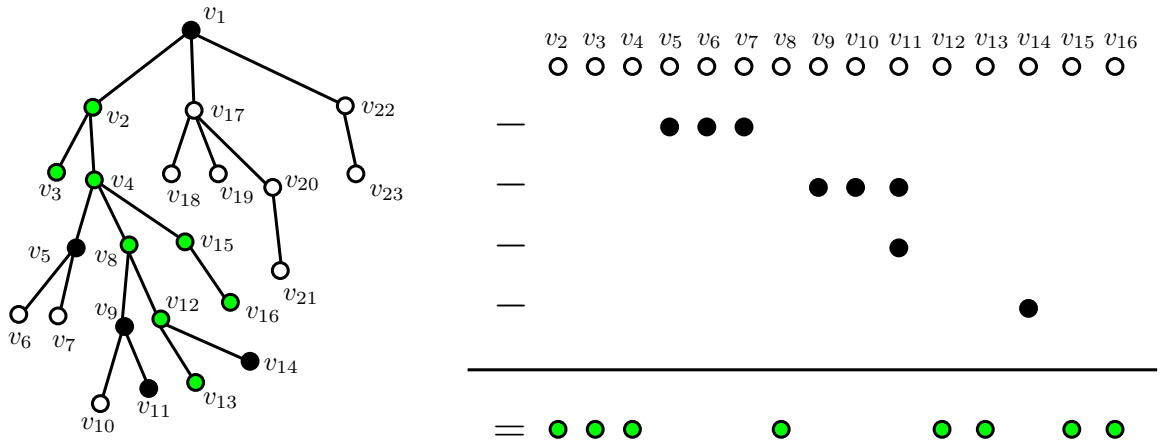



Figure 5: Left: a sample tree $T$. A possible Euler tour of $T$ is $v_1, v_2, \ldots, v_{23}$. We delete the vertices $\{v_1, v_5, v_9, v_{11}, v_{14}\}$ from $T$, and the connected component containing $v_2$ is marked as green. Right: the green component is a big interval (corresponding to the subtree of $v_2$) subtracting 4 smaller intervals (subtrees of $v_5, v_9, v_{11}$ and $v_{14}$), thus the union of $\leq 5$ intervals.

### 5.1.2 Euler Tours

For a tree $T$ rooted at $r \in V$, we perform a depth-first search on $T$ starting at $r$, and record every vertex at the first time it is encountered. The sequence of encountered vertices is called the *Euler tour* of $T$, denoted as $\mathrm{ET}(T)$. The Euler tour has a nice property, namely that every subtree of $T$ rooted at $x \in V(T)$ corresponds to an interval of the sequence $\mathrm{ET}(T)$. As a corollary, if we remove $d$ vertices $D$ from $T$, every connected component in $T - D$ (which is a smaller tree) corresponds to the union of $O(d)$ such intervals. (See Fig. 5 for an illustration.)

**Lemma 5.2.** *Let $D$ be a subset of $V(T)$ such that $|D| \leq d$, $S$ be any connected component of $T - D$, then $S$ is the union of $O(d)$ intervals of $\mathrm{ET}(T)$. Moreover these intervals can be found in $O(d \log d)$ time.*

*Proof.* Let $I(x)$ denote the interval of $\mathrm{ET}(T)$ corresponding to the subtree rooted at $x$. Let $h$ be the highest vertex in $S$, then $S = I(h) \setminus \bigcup_{f \in D} I(f)$, which is a big interval subtracting $d$ smaller intervals. By sorting the endpoints of $\{I(f) : f \in D \cup \{h\}\}$, we can express $S$ as the union of $O(d)$ intervals. □

## 5.2 Preprocessing Algorithm

In the preprocessing algorithm, (for each path $V = U_1, U_2, \ldots, U_p$ of the hierarchy tree,) we prune the trees and construct auxiliary data structures $H, E'$ as follows.

**Pruning the trees.** Recall that we consider the trees in $\mathcal{T} = \bigcup_{i=1}^{p} \mathcal{T}_{U_{i+1}}(U_i)$, and we have a distance parameter $\rho$. We prune off vertices of large depth in every tree in $\mathcal{T}$. For every $T \in \mathcal{T}$ and $v \in V(T)$, if $\mathrm{dep}_T(v) > (2k-1)\rho$, we delete $v$ from $T$. There are two reasons to perform this step:

- If for some $u \in V$, $\delta(u, v) \leq \rho$ and $u, v$ are covered by $T$ (i.e. $\mathrm{dep}_T(u) + \mathrm{dep}_T(v) \leq (2k-1) \cdot \delta(u, v)$), then $\mathrm{dep}_T(v) \leq (2k-1)\rho$, hence the pruning would not affect any distance of $\leq \rho$;
- After the pruning, every tree in $\mathcal{T}$ has diameter at most $(4k-2)\rho$.

In the rest of this section, we assume that all trees in $\mathcal{T}$ are pruned.

**The auxiliary DAG $H$.** We list the trees as $\mathcal{T} = \{T_1, T_2, \ldots, T_{|\mathcal{T}|}\}$, and concatenate their Euler tours as a list $\Lambda = \mathrm{ET}(T_1) \circ \mathrm{ET}(T_2) \circ \cdots \circ \mathrm{ET}(T_{|\mathcal{T}|})$. Recall that $|\mathcal{T}| = O(n)$ and every vertex appears in $O(h \log^2 n)$ trees, where $h = O(\log n / \log d)$. Therefore $|\Lambda| = O(nh \log^2 n)$ and $\alpha(|\Lambda|) = O(\alpha(n))$. Let $\mathcal{I}$ be an $O(\alpha(n))$-covering set of $\Lambda$, so every interval of $\Lambda$ can be expressed as the union of $O(\alpha(n))$ intervals in $\mathcal{I}$. We make two copies $\mathcal{I}_1, \mathcal{I}_2$ of $\mathcal{I}$, two copies $\mathcal{T}_1, \mathcal{T}_2$ of $\mathcal{T}$, and one copy $V_1$ of $V$. For $I \in \mathcal{I}$, let $I_1, I_2$ be its copies in $\mathcal{I}_1, \mathcal{I}_2$ respectively; $T_1, T_2, v_1$ are similarly defined.

We define a DAG $H$ with $V(H) = \mathcal{I}_1 \cup \mathcal{T}_1 \cup V_1 \cup \mathcal{T}_2 \cup \mathcal{I}_2$, and $E(H)$ defined as follows (where $(a \to b)$ denotes a directed edge from $a$ to $b$):

1. Let $I \in \mathcal{I}$, $T \in \mathcal{T}$, if there is an edge from some vertex in $I$ to some vertex in $T$ with weight $\le \rho$, then we have edges $(I_1 \to T_1)$ and $(T_2 \to I_2)$ in $E(H)$;
2. Let $T \in \mathcal{T}, u \in V(T)$ (after the pruning), then we have edges $(T_1 \to u_1)$ and $(u_1 \to T_2)$ in $E(H)$.

In the query algorithm, we use the graph $H$ to capture the paths only "involved" with *unaffected trees*, which are trees that do not intersect $D$ (see Definition 5.5). Therefore, we need to remove $\mathcal{T}_1^* \cup \mathcal{T}_2^*$ from $H$, where $\mathcal{T}^*$ is the set of affected trees, and $\mathcal{T}_1^*, \mathcal{T}_2^*$ are copies of $\mathcal{T}^*$ in $\mathcal{T}_1, \mathcal{T}_2$ respectively. Suppose we can upper bound the number of affected trees as $|\mathcal{T}^*| \le K$. We are interested in the following kind of queries on $H$: "Given $I_1 \in \mathcal{I}_1, I_2 \in \mathcal{I}_2$, can $I_1$ reach $I_2$ in $H - \mathcal{T}_1^* - \mathcal{T}_2^*$?" We claim that, since the depth of $H$ is a constant, such queries can be answered efficiently.

**Lemma 5.3.** *Let $H = (V, E)$ be a DAG, $V' \subseteq V$, $n = |V|$, $x, y$ be integers and $q = \frac{(x+y)^{x+y+1}}{x^x y^y} \log n$. Suppose every path in $H$ contains at most $x$ vertices in $V'$. We can build a data structure of size $O(qn^2)$ which, given a subset $D \in \binom{V'}{\le y}$ and two vertices $u, v \in V$, in $O(q)$ query time, outputs 1 if $u$ can reach $v$ in $H - D$ and 0 otherwise. With high probability over the randomized preprocessing algorithm, the data structure is correct on every query.*

The proof uses a clever trick of [34], which was inspired by color-coding [6].

Let $\mathcal{U} = \{1, 2, \ldots, n\}$, $\mathcal{S}$ be a family of subsets of $\mathcal{U}$. We say $\mathcal{S}$ is an $(x, y)$-*family* if for every $X \in \binom{\mathcal{U}}{x}$, $Y \in \binom{\mathcal{U}}{y}$ such that $X \cap Y = \emptyset$, there is a set $S \in \mathcal{S}$ such that $X \subseteq S$ and $Y \cap S = \emptyset$.

Fix $X, Y$, we randomly sample a subset $S$ of $\mathcal{U}$ by picking every element w.p. $\frac{x}{x+y}$. Then the set $S$ satisfies the condition that $X \subset S$ and $Y \cap S = \emptyset$ w.p. $p = \frac{x^x y^y}{(x+y)^{x+y}}$. By a union bound over all $X, Y$'s, if we sample $O(p^{-1} \log(n^{x+y})) = O\Big(\frac{(x+y)^{x+y+1}}{x^x y^y} \log n\Big)$ such sets $S$, we obtain an $(x, y)$-family with high probability.

*Remark* 5.4. The above construction of $(x, y)$-family can be derandomized by [48, Theorem 14]. In our regime where $x = 2$ and $y = \mathrm{polylog}(n)$, the randomized construction contains $O(y^3 \log n)$ sets, while the deterministic construction contains $O(y^3 \log^3 n)$ sets, slightly worse than the randomized construction. Below we will still use the randomized construction.

*Proof of Lemma 5.3.* Let $\mathcal{S}$ be an $(x, y)$-family of $V'$. For every $S \in \mathcal{S}$, we store a reachability matrix of the induced subgraph $H[S \cup (V \setminus V')]$. On a query $(u, v, D)$, the algorithm outputs 1 if and only if there is some $S \in \mathcal{S}$ such that $S \cap D = \emptyset$ and $u$ can reach $v$ in $H[S \cup (V \setminus V')]$.

The correctness of the algorithm follows directly from the definition of $(x, y)$-family. If $u$ does not reach $v$ in $H - D$, then for any $S$ such that $S \cap D = \emptyset$, $u$ does not reach $v$ in $H[S \cup (V \setminus V')]$. If $u$ can reach $v$ in $H - D$, let $P$ be the vertices in some specific path from $u$ to $v$ in $H - D$, then

$|P \cap V'| \leq x$ by hypothesis. By the definition of $(x, y)$-family, there is a set $S_0 \in \mathcal{S}$ such that $P \cap V' \subseteq S_0$ and $D \cap S_0 = \emptyset$, and the algorithm detects that $u$ can reach $v$ in $G[S_0 \cup (V \setminus V')]$. □

Let $x = 2, y = 2K, V' = \mathcal{T}_1 \cup \mathcal{T}_2$ and $q = \frac{(x+y)^{x+y+1}}{x^x y^y} \log |\Lambda| = O(K^3 \log n)$, Lemma 5.3 implies that we can maintain $H$ in $O(q \cdot |\Lambda|^2)$ space and answer the above queries in $O(q)$ time.

**The auxiliary table $E'$.** Besides the main structure $H$, we also need to store a table $E'$, specified as a subset of $\Lambda \times \Lambda$. For every $u, v \in V$ such that $u = v$ or $(u, v)$ is an edge with weight $\leq \rho$ in $E$, for every occurrences $u', v'$ of $u, v$ in $\Lambda$ respectively, there is an item $(u', v') \in E'$. There are no other items in $E'$.

Since every vertex occurs $O(h \log^2 n)$ times in $\Lambda$, we have $|E'| = O(mh^2 \log^4 n)$. We store $E'$ by a 2D range search structure [7] of size $O(mh^2 \log^5 n)$ such that given intervals $I_1, I_2$ of $\Lambda$, it can be queried if $E' \cap (I_1 \times I_2) = \emptyset$ in $O(\log \log n)$ time.

## 5.3 Query Algorithm

Suppose we are given $u, v \in V$, $D \in \binom{V}{\leq d}$ and $\rho$. As described earlier, we have already found a path $V = U_1, U_2, \ldots, U_p$ in the hierarchy tree where every vertex in $D$ has low pseudo-degree in every tree.

**Identifying affected trees.** We first identify the *affected trees* in $\mathcal{T}$. After the removal of $D$, these trees split into several subtrees, some of which are called *affected subtrees*, and the others are *ignored subtrees*. A precise definition is as follows:

**Definition 5.5.** A tree $T \in \mathcal{T}$ is an *affected tree* if $V(T) \cap D \neq \emptyset$. For each affected tree $T \in \mathcal{T}$, the removal of $D$ splits $T$ into several subtrees $T^{(1)}, T^{(2)}, \ldots, T^{(q)}$. A subtree $T^{(i)}$ is an *affected subtree* if it contains some trunk vertex of $T$; otherwise it is an *ignored subtree*. A vertex $v \in V$ is an *affected vertex* if it is in some affected subtree; otherwise it is an *unaffected vertex*.

*Remark* 5.6. It is possible that an unaffected vertex belongs to some ignored subtree.

**Lemma 5.7** (The Number of Affected (Sub)Trees)**.** *There are at most $O(dh \log^2 n)$ affected trees and $O(d^{c+2} h \log^4 n)$ affected subtrees.*

*Proof.* Recall that $p = O(h)$ is the depth of the high-degree hierarchy. Every vertex in $D$ is in at most $p \cdot 2e \ln^2 n$ trees, so at most $O(dh \log^2 n)$ trees can be affected. Every vertex in $D$ has pseudo-degree at most $s = O(d^{c+1} \log^2 n)$ in every tree, so these affected trees split into at most $O(d^{c+2} h \log^4 n)$ affected subtrees (and possibly many ignored subtrees). □

**The graph $R$.** During the query algorithm, we construct an unweighted graph $R$ whose vertex set is $V(R) = \{\{u\}, \{v\}\} \cup \mathcal{T}'$, where $\mathcal{T}'$ is the set of affected subtrees. We output YES if and only if $\{u\}$ and $\{v\}$ are connected in $R$.

For every $X, Y \in V(R)$, we use the following procedure to determine if there is an edge between $X$ and $Y$ in $E(R)$. We consider $X, Y$ as subsets of $V$. If $X$ is an affected subtree which belongs to the affected tree $T$, then by Lemma 5.2, we can write $X$ as the union of $O(d)$ intervals of $\mathrm{ET}(T)$. By Theorem 5.1, $X$ is the union of $O(d \cdot \alpha(n))$ intervals in $\mathcal{I}$. If $X = \{u\}$ or $X = \{v\}$ then $X$ is trivially an interval in $\mathcal{I}$. Similarly, we can also represent $Y$ as the union of $O(d \cdot \alpha(n))$ intervals in $\mathcal{I}$. If there are two intervals $I^1, I^2 \in \mathcal{I}$, where $I^1$ is in the representation of $X$ and $I^2$ is in the representation of $Y$, such that either $I^1_1$ can reach $I^2_2$ in $H - \mathcal{T}_1^* - \mathcal{T}_2^*$ or $E' \cap (I^1 \times I^2) \neq \emptyset$, then we

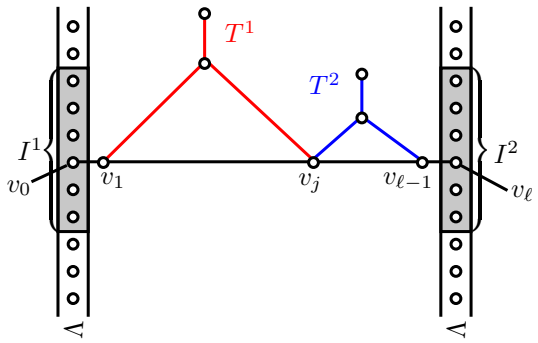



Figure 6: Proof of a).

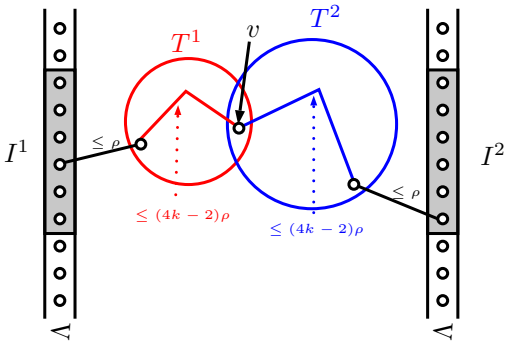



Figure 7: Proof of b).

insert an edge in $E(R)$ between $X$ and $Y$. Here $\mathcal{T}_1^*, \mathcal{T}_2^*$ denote the copies of affected trees in $\mathcal{T}_1, \mathcal{T}_2$ of $V(H)$ respectively.

The time complexity for the query algorithm is dominated by constructing $E(R)$. Since $|V(R)| = O(d^{c+2}h\log^4 n)$, and there are at most $K = O(dh\log^2 n)$ affected trees, the algorithm takes $O((d^{c+2}h\log^4 n)^2(d\cdot\alpha(n))^2(K^3\log n + \log\log n)) = O(d^{2c+9}\alpha^2(n)h^5\log^{15} n)$ time.

**Justification.** We justify the construction of the graph $R$. For $X, Y \in V(R)$, there should be an edge between $X$ and $Y$ if there is an *unaffected path* of length at most $\rho$ connecting them, defined as follows.

**Definition 5.8.** For $X, Y \subseteq V$, an *unaffected path* in $G - D$ connecting $X$ and $Y$ is a path $(v_0, \ldots, v_\ell)$ $(\ell \geq 0)$ in $G - D$ such that $v_0 \in X, v_\ell \in Y$, and $v_1, v_2, \ldots, v_{\ell-1}$ are unaffected vertices.

The following theorem justifies the definition of $E(R)$.

**Theorem 5.9.** *For $X, Y \in V(R)$:*

*a) If there is an unaffected path of length $\leq \rho$ connecting $X$ and $Y$, then $(X, Y) \in E(R)$.*
*b) If $(X, Y) \in E(R)$, then there is a path in $G - D$, which starts at some vertex in $X$, ends at some vertex in $Y$, and has length at most $(8k-2)\rho$.*

*Proof.* **Proof of a).** Let the path be $(v_0, v_1, \ldots, v_\ell)$ where $v_0 \in I^1, v_\ell \in I^2$, and $I^1, I^2$ are intervals in the representation of $X, Y$ respectively. If $\ell \leq 1$, then we have $(I^1 \times I^2) \cap E' \neq \emptyset$. If $\ell > 1$, let $v_j$ be the vertex in $v_1, v_2, \ldots, v_{\ell-1}$ with the highest level, and $i = l(v_j)$ be its level. Then the path does not intersect $U_{i+1}$. Let $T^1 = T_i(v_j, v_1)$ be the tree in $\mathcal{T}_{U_{i+1}}(U_i)$ which approximates the distance $\delta_{G-U_{i+1}}(v_j, v_1)$, then $\mathrm{dep}_{T^1}(v_1) + \mathrm{dep}_{T^1}(v_j) \leq (2k-1)\rho$. Therefore $v_1$ and $v_j$ are not pruned in $T^1$. If $T^1$ is an affected tree, then since $v_j$ is an unaffected vertex, it must lie in some ignored subtree of $T^1$, but this contradicts the fact that $v_j \in \mathrm{Trunk}(T^1)$. Therefore $T^1$ is not an affected tree. Similarly let $T^2 = T_i(v_j, v_{\ell-1})$, then $\mathrm{dep}_{T^2}(v_{\ell-1}) + \mathrm{dep}_{T^2}(v_j) \leq (2k-1)\rho$, and $T^2$ is not an affected tree. We conclude that there is a path $I_1^1 \to T_1^1 \to (v_j)_1 \to T_2^2 \to I_2^2$ in $H - \mathcal{T}_1^* - \mathcal{T}_2^*$.

**Proof of b).** Suppose that the interval $I^1$ is in the representation of $X$, the interval $I^2$ is in the representation of $Y$, and $I^1, I^2$ contributes to the edge $(X, Y)$. If $(I^1 \times I^2) \cap E' \neq \emptyset$, then either $I^1 \cap I^2 \neq \emptyset$ or there is an edge $(u, v) \in E$ of length $\leq \rho$ such that $u \in I^1$ and $v \in I^2$. In either case the lemma follows. On the other hand, if $I_1^1$ can reach $I_2^2$ in $H - \mathcal{T}_1^* - \mathcal{T}_2^*$, and the corresponding path in $H - \mathcal{T}_1^* - \mathcal{T}_2^*$ is $I_1^1 \to T_1^1 \to v_1 \to T_2^2 \to I_2^2$, then $T^1$ and $T^2$ are unaffected trees. Consider the following path $p$, which starts from $I^1$, goes to an adjacent vertex in $T^1$ by an edge of weight

$\leq \rho$, walks along $T^1$ to reach $v$, walks along $T^2$ to reach a vertex adjacent to $I^2$ by an edge of weight $\leq \rho$, then goes to $I^2$. Since every tree has diameter at most $(4k-2)\rho$, the length of $p$ is at most $(8k-2)\rho$. Since $T^1$ and $T^2$ are unaffected trees, $p$ avoids $D$, and the lemma follows. □

Given Theorem 5.9, it is easy to prove that our algorithm achieves a stretch of $O(k \cdot |V(R)|) = O(d^{c+2}h\log^5 n)$.

**Theorem 5.10** (Correctness)**.** *There is some $A = O(d^{c+2}h\log^5 n)$ such that for $u, v \in V$, $D \in \binom{V}{\leq d}$:*

*a) If $\delta_{G-D}(u,v) \leq \rho$, then the algorithm outputs* YES.
*b) If the algorithm outputs* YES*, then $\delta_{G-D}(u,v) \leq \rho \cdot A$.*

*Proof.* **Proof of a).** Suppose $p : (u = w_0, w_1, \ldots, w_{\ell-1}, w_\ell = v)$ is a path from $u$ to $v$ in $G-D$ with length at most $\rho$. For $1 \leq i < \ell$, let $t_i$ be any affected subtree that $w_i$ lies in; if $w_i$ is an unaffected vertex then set $t_i = \emptyset$. Let $a(i)$ $(i \geq 1)$ be the $i$-th index such that $t_{a(i)} \neq \emptyset$, and $q$ be the maximum index such that $a(q)$ is defined. (For example, $t_j = \emptyset$ for every $1 \leq j < a(1)$ or $a(q) < j < \ell$, but $t_{a(1)} \neq \emptyset, t_{a(q)} \neq \emptyset$.) Artificially we define $t_0 = \{u\}, a(0) = 0, t_\ell = \{v\}$ and $a(q+1) = \ell$. Then for every $0 \leq i \leq q$, we have an unaffected path $(w_{a(i)}, w_{a(i)+1}, \ldots, w_{a(i+1)})$ of length $\leq \rho$ connecting $t_{a(i)}$ and $t_{a(i+1)}$, thus $(t_{a(i)}, t_{a(i+1)}) \in E(R)$ by Theorem 5.9 a). We conclude that $\{u\}$ and $\{v\}$ are connected in $R$. Therefore the algorithm returns YES.

**Proof of b).** Suppose the algorithm returns YES. Then there is a simple path $\{u\} = t_0 \to t_1 \to \cdots \to t_\ell = \{v\}$ in $R$. By Theorem 5.9 b), there are vertices $u = u_0, v_1, u_1, \ldots, v_{\ell-1}, u_{\ell-1}, v_\ell = v$ such that:

- For every $1 \leq i < \ell$, $v_i, u_i \in t_i$.
- For every $0 \leq i < \ell$, $\delta_{G-D}(u_i, v_{i+1}) \leq (8k-2)\rho$.

Since each tree has diameter at most $(4k-2)\rho$, we can add that:

- For every $1 \leq i < \ell$, $\delta_{G-D}(v_i, u_i) \leq (4k-2)\rho$.

Therefore $\delta_{G-D}(u,v) \leq \ell \cdot (8k-2)\rho + (\ell-1)\cdot(4k-2)\rho = ((12k-4)\ell - 4k + 2)\,\rho$. Since $\ell \leq |V(R)| = O(d^{c+2}h\log^4 n)$, we have $\delta_{G-D}(u,v) \leq O(d^{c+2}h\log^5 n)\cdot\rho$. □

*Remark* 5.11. By investigating the proofs of Theorem 5.9 b) and Theorem 5.10 b), we can retrieve a path from $u$ to $v$ in $G-D$ of length $O(d^{c+2}h\log^5 n)\cdot\rho$ in $O(\ell)$ additional time, where $\ell$ is the number of nodes in the retrieved path.

The space complexity of our oracle is dominated by the $O(n^{1/c}q|\Lambda|^2)$ term, therefore our VSDO has

$$\begin{aligned} \text{space complexity} &\quad O(n^{2+1/c}d^3\log^{16} n\log(nW)/\log^5 d), \\ \text{query complexity} &\quad O(d^{2c+9}\alpha^2(n)\log^{20} n\log\log(nW)/\log^5 d), \\ \text{stretch} &\quad O(d^{c+2}\log^6 n/\log d). \end{aligned}$$

## Acknowledgments

We are grateful to anonymous reviewers for helpful comments, bringing [59] to our attention, and pointing out the recent work [48] that allows us to derandomize the oracle in Section 5. We would like to thank Thatchaphol Saranurak for providing an early manuscript of [68], and Zhijun Zhang for helpful comments on a draft version of this paper.

## References


[1] Ittai Abraham, Shiri Chechik, and Sebastian Krinninger. Fully dynamic all-pairs shortest paths with worst-case update-time revisited. In *Proc. 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 440–452, 2017.

[2] Ittai Abraham, Shiri Chechik, and Kunal Talwar. Fully dynamic all-pairs shortest paths: Breaking the $O(n)$ barrier. In *Proc. Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, (APPROX/RANDOM)*, volume 28 of *LIPIcs*, pages 1–16, 2014.

[3] 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.

[4] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. On finding lowest common ancestors in trees. In *Proc. 5th Annual ACM Symposium on Theory of Computing (STOC)*, pages 253–265, 1973.

[5] Noga Alon and Baruch Schieber. Optimal preprocessing for answering on-line product queries. Technical Report 71/87, Tel Aviv University, 1987.

[6] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, July 1995.

[7] Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *Proc. 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 198–207, 2000.

[8] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete and Computational Geometry*, 9:81–100, 1993.

[9] 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.

[10] Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *Proc. 4th Latin American Theoretical Informatics Symposium (LATIN)*, volume 1776 of *LNCS*, pages 88–94, 2000.

[11] Aaron Bernstein. Fully dynamic $(2 + \epsilon)$ approximate all-pairs shortest paths with fast query and close to linear update time. In *Proc. 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 693–702, 2009.

[12] Aaron Bernstein and David Karger. Improved distance sensitivity oracles via random sampling. In *Proc. 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 34–43, 2008.

[13] Aaron Bernstein and David Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *Proc. 41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 101–110, 2009.

[14] Davide Bilò, Keerti Choudhary, Luciano Gualà, Stefano Leucci, Merav Parter, and Guido Proietti. Efficient oracles and routing schemes for replacement paths. In *Proc. 35th Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 96 of *LIPIcs*, pages 13:1–13:15, 2018.

[15] Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Multiple-edge-fault-tolerant approximate shortest-path trees. In *Proc. 33rd Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 47 of *LIPIcs*, pages 18:1–18:14, 2016.

[16] Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Fault-tolerant approximate shortest-path trees. *Algorithmica*, 80(12):3437–3460, 2018.

[17] Greg Bodwin, Michael Dinitz, Merav Parter, and Virginia Vassilevska Williams. Optimal vertex fault tolerant spanners (for fixed stretch). In *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1884–1900, 2018.

[18] Greg Bodwin, Fabrizio Grandoni, Merav Parter, and Virginia Vassilevska Williams. Preserving distances in very faulty graphs. In *Proc. 44th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 80 of *LIPIcs*, pages 73:1–73:14, 2017.

[19] Greg Bodwin and Shyamal Patel. A trivial yet optimal solution to vertex fault tolerant spanners. In *Proc. 38th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 541–543, 2019.

[20] Glencora Borradaile, Seth Pettie, and Christian Wulff-Nilsen. Connectivity oracles for planar graphs. In *Proc. 13th Scandinavian Symposium and Workshop on Algorithm Theory (SWAT)*, volume 7357 of *LNCS*, pages 316–327, 2012.

[21] Panagiotis Charalampopoulos, Shay Mozes, and Benjamin Tebeka. Exact distance oracles for planar graphs with failing vertices. In *Proc. 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2110–2123, 2019.

[22] S. Chechik, M. Langberg, David Peleg, and L. Roditty. Fault-tolerant spanners for general graphs. In *Proc. 41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 435–444, 2009.

[23] Shiri Chechik. *Fault-tolerant structures in graphs*. PhD thesis, Weizmann Institute of Science, June 2012.

[24] Shiri Chechik and Sarel Cohen. Distance sensitivity oracles with subcubic preprocessing time and fast query time. In *Proc. 52nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 1375–1388, 2020.

[25] Shiri Chechik, Sarel Cohen, Amos Fiat, and Haim Kaplan. $(1+\epsilon)$-approximate $f$-sensitive distance oracles. In *Proc. 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1479–1496, 2017.

[26] Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. $f$-sensitivity distance oracles and routing schemes. *Algorithmica*, 63(4):861–882, 2012.

[27] Keerti Choudhary. An optimal dual fault tolerant reachability oracle. In *Proc. 43rd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 55 of *LIPIcs*, pages 130:1–130:13, 2016.

[28] Rezaul Alam Chowdhury and Vijaya Ramachandran. Improved distance oracles for avoiding link-failure. In *Proc. 13th International Symposium on Algorithms and Computation (ISAAC)*, volume 2518 of *LNCS*, pages 523–534, 2002.

[29] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

[30] Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM*, 51(6):968–992, 2004.

[31] Camil Demetrescu and Giuseppe F. Italiano. Fully dynamic all pairs shortest paths with real edge weights. *Journal of Computer and System Sciences*, 72(5):813–837, 2006.

[32] Camil Demetrescu and Mikkel Thorup. Oracles for distances avoiding a link-failure. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 838–843, 2002.

[33] Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM Journal of Computing*, 37(5):1299–1318, 2008.

[34] Michael Dinitz and Robert Krauthgamer. Fault-tolerant spanners: Better and simpler. In *Proc. 30th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 169–178, 2011.

[35] Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In *Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 506–515, 2009.

[36] Ran Duan and Seth Pettie. Connectivity oracles for failure prone graphs. In *Proc. 42nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 465–474, 2010.

[37] Ran Duan and Seth Pettie. Connectivity oracles for graphs subject to vertex failures. In *Proc. 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 490–509, 2017.

[38] Ran Duan and Tianyi Zhang. Improved distance sensitivity oracles via tree partitioning. In *Proc. 15th International Symposium on Algorithms and Data Structures (WADS)*, volume 10389 of *LNCS*, pages 349–360, 2017.

[39] P. Erdős. Extremal problems in graph theory. In *Proceedings of the Symposium on Theory of Graphs and its Applications*, pages 29–36, 1964.

[40] David Gibb, Bruce M. Kapron, Valerie King, and Nolan Thorn. Dynamic graph connectivity with improved worst case update time and sublinear space. *CoRR*, abs/1509.06464, 2015.

[41] Fabrizio Grandoni and Virginia Vassilevska Williams. Improved distance sensitivity oracles via fast single-source replacement paths. In *Proc. 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 748–757, 2012.

[42] Manoj Gupta and Aditi Singh. Generic single edge fault tolerant exact distance oracle. In *Proc. 45th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 107 of *LIPIcs*, pages 72:1–72:15, 2018.

[43] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal of Computing*, 13(2):338–355, 1984.

[44] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs. In *Proc. 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 674–683, 2014.

[45] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A subquadratic-time algorithm for decremental single-source shortest paths. In *Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1053–1072, 2014.

[46] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Dynamic approximate all-pairs shortest paths: Breaking the $O(mn)$ barrier and derandomization. *SIAM Journal of Computing*, 45(3):947–1006, 2016.

[47] Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proc. 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1131–1142, 2013.

[48] Karthik C. S. and Merav Parter. Deterministic replacement path covering. *CoRR*, abs/2008.05421, 2020.

[49] Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proc. 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 81–91, 1999.

[50] François Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. 39th International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 296–303, 2014.

[51] Merav Parter. Dual failure resilient BFS structure. In *Proc. 2015 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 481–490, 2015.

[52] Merav Parter. Fault-tolerant logical network structures. *Bulletin of the EATCS*, 118, 2016.

[53] Merav Parter. Vertex fault tolerant additive spanners. *Distributed Computing*, 30(5):357–372, 2017.

[54] Merav Parter and David Peleg. Fault tolerant BFS structures: A reinforcement-backup tradeoff. In *Proc. 27th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 264–273, 2015.

[55] Merav Parter and David Peleg. Sparse fault-tolerant BFS structures. *ACM Transactions on Algorithms*, 13(1):11:1–11:24, October 2016.

[56] Merav Parter and David Peleg. Fault-tolerant approximate BFS structures. *ACM Transactions on Algorithms*, 14(1):10:1–10:15, January 2018.

[57] Mihai Pătraşcu and Mikkel Thorup. Planning for fast connectivity updates. In *Proc. 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 263–271, 2007.

[58] Hanlin Ren. Improved distance sensitivity oracles with subcubic preprocessing time. In *Proc. 28th European Symposium on Algorithms (ESA)* , volume 173 of *LIPIcs*, pages 79:1–79:13, 2020.

[59] Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 261–272, 2005.

[60] Liam Roditty and Uri Zwick. Dynamic approximate all-pairs shortest paths in undirected graphs. In *Proc. 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 499–508, 2004.

[61] Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, October 2011.

[62] Piotr Sankowski. Subquadratic algorithm for dynamic shortest distances. In *Proc. 11th International Computing and Combinatorics Conference (COCOON)* , volume 3595 of *LNCS*, pages 461–470, 2005.

[63] Andrew James Stothers. *On the complexity of matrix multiplication.* PhD thesis, The University of Edinburgh, 2010.

[64] Mikkel Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In *Proc. 9th Scandinavian Symposium and Workshop on Algorithm Theory (SWAT)*, volume 3111 of *LNCS*, pages 384–396, 2004.

[65] Mikkel Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proc. 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 112–119, 2005.

[66] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM*, 52(1):1–24, 2005.

[67] Jan van den Brand and Danupon Nanongkai. Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time. In *Proc. 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 436–455, 2019.

[68] Jan van den Brand and Thatchaphol Saranurak. Sensitive distance and reachability oracles for large batch updates. In *Proc. 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 424–435, 2019.

[69] Zhengyu Wang. An improved randomized data structure for dynamic graph connectivity. *CoRR*, abs/1510.04590, 2015.

[70] Oren Weimann and Raphael Yuster. Replacement paths and distance sensitivity oracles via fast matrix multiplication. *ACM Transactions on Algorithms*, 9(2):14:1–14:13, March 2013.

[71] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In Howard J. Karloff and Toniann Pitassi, editors, *Proc. 44th Annual ACM Symposium on Theory of Computing (STOC)*, pages 887–898, 2012.

[72] Andrew C. Yao. Space-time tradeoff for answering range queries (extended abstract). In *Proc. 14th Annual ACM Symposium on Theory of Computing (STOC)*, pages 128–136, 1982.

# A Proof of Theorem 2.2

We start with a randomized construction. We construct a sequence of nested subsets $S = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_k = \emptyset$, where each $A_i$ $(1 \leq i < k)$ is constructed by independently sampling each vertex in $A_{i-1}$ w.p. $n^{-1/k}$. Let $w \in S$, then there is some $0 \leq i < k$ such that $w \in A_i \setminus A_{i+1}$. We define a *cluster* $C(w)$ around $w$ as follows:

$$C(w) = \{v \in V : \delta(v, w) < \delta(v, A_{i+1})\}.$$

That is, if $v$ is closer to $w$ than to every vertex in $A_{i+1}$, then $v \in C(w)$. Let $T(w)$ be the shortest path tree rooted at $w$ spanning $C(w)$. The tree cover is $\mathcal{T}(S) = \{T(w) : w \in S\}$.

It is easy to see that the tree $T(w)$ only contains vertices in $C(w)$. Actually, let $v \in C(w)$, $v'$ be a vertex on the shortest path from $w$ to $v$, then $v' \in C(w)$, since

$$\begin{aligned}\delta(v', w) &= \delta(v, w) - \delta(v, v') \\ &< \delta(v, A_{i+1}) - \delta(v, v') \\ &\leq \delta(v', A_{i+1}).\end{aligned}$$

For every vertex $v \in V$, we also define a *bunch* $B(v)$ as follows. For $w \in A_i \setminus A_{i+1}$, if $\delta(v, w) < \delta(v, A_{i+1})$, then $w$ is in the bunch $B(v)$. (For any $v$ and $i$, let the vertex in $A_i$ closest to $v$ be $u$, then $u$ must be in $A_j \setminus A_{j+1}$ for some $j \geq i$, so $u \in B(v)$.) It is easy to check that

$$B(v) = \{w \in S : v \in C(w)\}.$$

Now we derandomize the construction of $A_i$ and justify Definition 2.1 c). (That is, every vertex is in $\leq kn^{1/k}(\ln n + 1)$ trees.) For every vertex $v \in V$, since $v$ is only in the trees rooted in $B(v)$, it suffices to prove that $|B(v)| \leq kn^{1/k}(\ln n + 1)$. Suppose we have constructed $A_i$ and want to construct $A_{i+1}$ now. For $v \in V$, let $N_{i+1}(v)$ be the set of the $n^{1/k}(\ln n + 1)$ closest vertices to $v$ in $A_i$. By [66, Lemma 3.6], a hitting set $A_{i+1}$ of the family $\{N_{i+1}(v) : v \in V\}$ can be found in polynomial time with $|A_{i+1}| \leq n^{-1/k}|A_i|$, and this finishes the construction of $A_{i+1}$. For each vertex $v \in V$ and level $i$, since $A_{i+1} \cap N_{i+1}(v) \neq \emptyset$, we have that $|B(v) \cap (A_i \setminus A_{i+1})| \leq n^{1/k}(\ln n + 1)$. It follows that $|B(v)| \leq kn^{1/k}(\ln n + 1)$.

It remains to justify Definition 2.1 b). That is, for every $u \in S$ and $v \in V$, there is some $w \in S$ such that $u, v \in V(T(w))$, and $\mathrm{dep}_{T(w)}(u) + \mathrm{dep}_{T(w)}(v) \leq (2k-1)\delta(u, v)$.

- If $u \in B(v)$, then we can pick $w = u$, and $\mathrm{dep}_{T(w)}(u) + \mathrm{dep}_{T(w)}(v) = \delta(u, v)$.
- Otherwise, assume $u \in A_{i_0} \setminus A_{i_0+1}$, and let $w_1$ be the vertex in $A_{i_0+1}$ closest to $v$. Then $\delta(w_1, v) \leq \delta(u, v)$, thus $\delta(w_1, u) \leq 2\delta(u, v)$. We also have that $w_1 \in B(v)$, i.e. $v \in C(w_1)$. If $w_1 \in B(u)$, then we can pick $w = w_1$, and $\mathrm{dep}_{T(w)}(u) + \mathrm{dep}_{T(w)}(v) \leq 3\delta(u, v)$.
- Otherwise ($w_1 \notin B(u)$), assume $w_1 \in A_{i_1} \setminus A_{i_1+1}$, and let $w_2$ be the vertex in $A_{i_1+1}$ closest to $u$. Then $\delta(w_2, u) \leq \delta(w_1, u) \leq 2\delta(u, v)$, thus $\delta(w_2, v) \leq 3\delta(u, v)$. We also have that $w_2 \in B(u)$, i.e. $u \in C(w_2)$. If $w_2 \in B(v)$, then we can pick $w = w_2$, and $\mathrm{dep}_{T(w)}(u) + \mathrm{dep}_{T(w)}(v) \leq 5\delta(u, v)$.
- Otherwise ($w_2 \notin B(v)$) ...
- Repeat this procedure until we find a tree $T(w)$ containing both $u$ and $v$.

The levels $i_0, i_1, \ldots$ are strictly increasing, so we reach level $k-1$ in $O(k)$ time (if we did not terminate before). For every $v \in V$, we have $A_{k-1} \subseteq B(v)$, so the procedure indeed terminates in $O(k)$ time. It is easy to see that the stretch is at most $2k-1$.

# B Additional Figures and Tables

| Notation | Meaning | Remarks |
|---|---|---|
| $\circ$ | path/sequence concatenation operator | |
| $w_H(u,v)$ | the weight of edge $(u,v)$ in $H$ | We omit $H$ when $H = G$ is the input graph. |
| $\delta_H(u,v)$ | the distance between $u$ and $v$ in $H$ | |
| $\pi_H(u,v)$ | the shortest $u$-$v$ path in $H$ | |
| $\delta_H(u,S)$ | $\min\{\delta_H(u,v) : v \in S\}$ | |
| $G[S]$ | the subgraph of $G$ induced by $S$ | $S \subseteq V$. |
| $P[u,v]$<br>$P[u,v)$<br>$P(u,v]$<br>$P(u,v)$ | the portion between $u$ and $v$ of path $P$<br>$P[u,x_{\ell-1}]$<br>$P[x_1,v]$<br>$P[x_1,x_{\ell-1}]$ | Assume $P = (x_0,x_1,\ldots,x_{\ell-1},x_\ell)$ where $x_0 = u, x_\ell = v$; These notations sometimes emphasize the *direction* of the path. |
| $l(v)$ | the level of $v$, or the largest $i$ such that $v \in U_i$ | |
| $G_\ell$ | the subgraph of $G$ induced by vertices with level $\leq \ell$ | |
| $\mathcal{P}_\ell(x,y)$ | the $x$-$y$ path in $\mathcal{T}_{U_{\ell+1}}(U_\ell)$ guaranteed by Corollary 2.8 | $x \in U_\ell \setminus U_{\ell+1}, y \in V \setminus U_{\ell+1}$. |
| $T_\ell(x,y)$ | the tree in $\mathcal{T}_{U_{\ell+1}}(U_\ell)$ that contains $\mathcal{P}_\ell(x,y)$ | |

Table 1: Notation in this paper

| failure | # fault | size | query time | stretch | ref | remarks |
|---|---|---|---|---|---|---|
| edge | 1 | $O(n^2 \log n)$ | $O(\log n)$ | 1 | [32] | directed |
| edge | 1 | $O(n^2 \log n)$ | $O(1)$ | 1 | [28] | directed |
| vertex | 1 | $O(n^2 \log n)$ | $O(1)$ | 1 | [12, 13, 33] | directed |
| vertex | 1 | $O(n^2)$ | $O(1)$ | 1 | [38] | directed |
| vertex | 1 | $O(k^5 \epsilon^{-4} n^{1+1/k} \log^3 n)$ | $O(k)$ | $(2k-1)(1+\epsilon)$ | [9] | unweighted |
| vertex | 2 | $O(n^2 \log^3 n)$ | $O(\log n)$ | 1 | [35] | directed |
| vertex | 2 | $O(n^2)$ | $O(1)$ | reachability | [27] | directed |
| edge | $d$ | $O(m)$ | $O(d \log^{2.5} n \log\log n)$ | connectivity | [57] | |
| edge | $d$ | $O(m \log\log n)$ | $O(d^2 \log\log n)$ | connectivity | [36] | |
| edge | $d$ | $O(m)$ | $O(d^2 \log^{\epsilon} n)$ | connectivity | [36] | |
| edge | $d$ | $O(n \log^2 n)$ | $O(d \log d \log\log n)$ | connectivity | [37] | |
| edge | $d$ | $O\big(dkn^{1+1/k} \log(nW)\big)$ | $O\big(d \log^2 n \log\log n \log\log(nW)\big)$ | $(8k-2)(d+1)$ | [26] | |
| edge | $d$ | $O(dn^2 \log^2 n)$ | $O(d^2 \log^2 n)$ | $2d+1$ | [15] | |
| edge | $d$ | $O\big(n^3 (\log n/\epsilon)^d (\log W/\log n)\big)$ | $O(d^4 \log\log W)$ | $1+\epsilon$ | [25] | |
| edge | $d$ | $O\big(n^2 (\log n/\epsilon)^d \cdot d \log W\big)$ | $O(d^5 \log n \log\log W)$ | $1+\epsilon$ | [25] | |
| vertex | $d$ | $O\big(d^{1-2/c} m n^{1/c - 1/(c\log(2d))} \log^2 n\big)$ | $O\big(d^{2c+4} \log^2 n \log\log n\big)$ | connectivity | [36] | $c \geq 1$ |
| vertex | $d$ | $O(m \log^6 n)$ | $O\big(d^2 \log d \log^2 n \log\log n\big)$ | connectivity | [37] | |
| vertex | $d$ | $O(n/r)^{d+1} \frac{\sqrt{ndr}}{d!} + O(n \log^2 n)$ | $O(d\sqrt{r} \log^2 n)$ | 1 | [21] | planar; $r \leq n/d$ |
| edge | $d$ | $O(Wn^{2+\mu} \log n)$ | $\tilde{O}(Wn^{2-\mu} d^2 + Wnd^{\omega})$ | 1 | [68] | directed; $\mu \in [0,1]$ |
| edge | $d$ | $O(n^2 \log n)$ | $O(d^{\omega})$ | reachability | [68] | directed |

Table 2: previous results[18]

| failure | # fault | size | query time | stretch | ref | remarks |
|---|---|---|---|---|---|---|
| vertex | $d$ | $n^{3+1/c} \cdot O\Big(\epsilon^{-1} \frac{\log^2 n \log(nW)}{\log d}\Big)^d$ | $O\Big(\frac{d^{2c+6} \log^{10} n}{\log^2 d}\Big)$ | $1+\epsilon$ | this paper | $c \geq 1$ |
| vertex | $d$ | $n^{3+1/c} \frac{\log W}{\log n} \cdot O\Big(\epsilon^{-1} \frac{\log^3 n}{\log d}\Big)^d$ | $O\Big(\frac{d^{2c+6} \log^{10} n \log\log W}{\log^2 d}\Big)$ | $1+\epsilon$ | this paper | $c \geq 1$ |
| vertex | $d$ | $n^{2+1/c} \frac{\log d}{\log n} \cdot O\Big(\epsilon^{-1} \frac{\log^2 n \log(nW)}{\log d}\Big)^{d+1}$ | $O\Big(\epsilon^{-1} \frac{d^{2c+6} \log^{11} n \log(nW)}{\log^2 d}\Big)$ | $1+\epsilon$ | this paper | $c \geq 1$ |
| vertex | $d$ | $n^{2+1/c} \frac{\log W \log d}{\log^2 n} \cdot O\Big(\epsilon^{-1} \frac{\log^3 n}{\log d}\Big)^{d+1}$ | $O\Big(\epsilon^{-1} \frac{d^{2c+6} \log^{12} n \log\log W}{\log^2 d}\Big)$ | $1+\epsilon$ | this paper | $c \geq 1$ |
| vertex | $d$ | $O\Big(\frac{n^{2+1/c} d^3 \log^{16} n \log(nW)}{\log^5 d}\Big)$ | $O\Big(\frac{d^{2c+9} \alpha^2(n) \log^{20} n \log\log(nW)}{\log^5 d}\Big)$ | $O\Big(\frac{d^{c+2} \log^6 n}{\log d}\Big)$ | this paper | $c \geq 1$ |

Table 3: our results

[18] Unless stated in "remark" column, all data structures work on weighted undirected graphs.