Approximate Distance Oracles Subject to Multiple Vertex Failures *

Ran Duan† Yong Gu‡ Hanlin Ren§

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 \( G \). This graph \( G \) with vertices in \( D \) removed. An oracle has stretch \( \alpha \) if it always holds that \( \delta_{G-D}(u,v) \leq \alpha \cdot \delta_{G}(u,v) \), where \( \delta_{G}(u,v) \) is the actual distance between \( u \) and \( v \) in \( G \).

In this paper we construct efficient VSDOs for any number \( d \) of failures. For any constant \( \epsilon > 0 \), Chechik et al. [21] designed a data structure that maintains \( O(d) \)-approximate shortest paths under \( d \) edge failures in an undirected graph, and Bilò et al. [10] improved the approximation ratio to \( 2d + 1 \). For any \( \epsilon > 0 \), Chechik et al. [20] designed a data structure that maintains \((1 + \epsilon)\)-approximate shortest paths under \( d \) edge failures in an undirected graph, with space complexity \( O(n^{2}(\log n)/\epsilon) \cdot d \log W \) and query time \( \log(n, d, \log W) \), where \( W \) is the ratio of the largest edge weight to the smallest edge weight. More related work can be found in Section 1.1.

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 [5, 21, 20], and also in Chechik’s PhD thesis [18].

In this paper we build efficient data structures that answer approximate distance queries under multiple vertex failures for general undirected graphs. This problem was addressed as an open problem in [5, 21, 20], and also in Chechik’s PhD thesis [18].

In their pioneering work, Demetrescu and Thorup [27] 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 [28] extends the structure to also handle one vertex failure, and improves the query time to \( O(1) \).


dedicated to the memory of our close friend, Wu-Yi Hsiang. His contributions to the field of algorithms and complexity have had a profound impact on our work and on the field as a whole.

---

*This work has been supported in part by the Zhongguancun Hailua Institute for Frontier Information Technology.
†Institute for Interdisciplinary Information Sciences, Tsinghua University.
‡Institute for Interdisciplinary Information Sciences, Tsinghua University.
§Institute for Interdisciplinary Information Sciences, Tsinghua University.

---

**We can also use dynamic connectivity structures with poly-logarithmic worst case update time [41, 35, 63] to handle \( d \) edge failures.**

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited
(u, v, D), we always have |D| ≤ 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 α if it always holds that δ_{G-D}(u, v) ≤ δ(u, v, D) ≤ α·δ_{G-D}(u, v), where δ_{G-D}(u, v) is the actual distance between u and v in G - D (i.e. G with D disabled), and δ(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].

1.1 More Related Work.

**Sensitivity oracles.** For the case of two vertex failures, Duan and Pettie [30] showed that exact distances in a directed weighted graph can be queried in time \(O(n^2 \log^3 n)\), and Choudhary [22] 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. [15] constructed a data structure that maintains connectivity under d vertex failures, and Charalamposoulos et al. [16] designed a data structure that answers exact distance queries under d vertex failures.

In a recent breakthrough, van den Brand and Saranurak [62] gave an oracle that handles an arbitrary number d of edge failures in directed graphs. Their oracle can answer reachability queries in \(O(d^3)\) time, and exact distance queries in \(n^{2+\Omega(1)}\) time (for small integer weights), where \(\omega < 2.3728639\) is the matrix-multiplication exponent [24, 57, 65, 44].

We summarize the sensitivity connectivity/distance oracles in the full version (Table 2 of Appendix C) of this paper.

**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. [17] gave the first construction of \(d\)-FT \((2k - 1)\)-spanners with \(O(d^2 k^{d+1} n^{1+1/k} \log^{1-1/k} n)\) edges. Subsequent papers [29, 12, 14] improved the number of edges to \(O(n^{1+1/k} d^{1-1/k})\), which is optimal assuming the girth conjecture of Erdős [34].

Besides FT spanners, there are many other kinds of fault-tolerant structures, e.g. [11, 45, 48, 49, 10, 13, 47, 50]. We refer the reader to the excellent survey of [46].

**Dynamic shortest path.** There are dynamic all-pairs shortest path structures handling vertex updates. Thorup [59] gave a fully dynamic all-pairs shortest path algorithm with worst-case update time \(O(n^{2.75})\), and Abraham et al. [1] gave a randomized worst-case update time bound \(O(n^{2+2/3})\). Recently, Brand and Nanongkai [61] 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. Using it we can construct a \(\tilde{O}(d^{-2} n^{1.863})\) query time oracle for vertex failures in undirected graphs by a simple reduction. Other fully or partial dynamic shortest path structures include [2, 6, 26, 40, 38, 39, 43, 55, 54, 58, 56].

1.2 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:

**Theorem 1.1.** 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 \(\tilde{O}(d^{c+1} \epsilon^{-1} \log \log W)\) and stretch \(1 + \epsilon\);

(b) space complexity \(\tilde{O}(n^{2+1/c} d^{3} \log(nW))\), query time \(\tilde{O}(d^{c+9} \log(nW))\) and stretch \(O(d^{c+2} \log^2 n)\).

Each oracle can be preprocessed in time polynomial in their space complexity. 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 \(n^{O(d)}\) space or \(\Omega(n)\) query time. Thus our results are the first of its kind.
1.3 A brief overview. In this section, we briefly introduce the ideas needed to construct the desired VSDOs.

The edge-sensitive distance oracle of [20]. Our first VSDO depends on [20] 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 $P_{ans}$ 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 $O(\epsilon^{-1} \log W)$ short segments. Consider a segment $X$ that contains some failed edges. If $P_{ans}$ does not go through $X$, then we can “preprocess” the graph $G - X$ and search for $P_{ans}$ in $G - X$. Otherwise, if $P_{ans}$ 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 $P_{ans}$ 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 polynomial of $d$. 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 $\epsilon \cdot |P_{ans}|$, by partitioning each path into sufficiently short segments.

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 [21, 20] and delete at most $d \cdot \Delta$ edges from it. Equivalently, we can define the set of intermediate vertices $H$ as those active vertices adjacent to some failure, then $|H| \leq d \cdot \Delta$ and we run the above query algorithm. However the techniques of [21, 20] do not seem to work for high-degree vertex failures. For example, techniques in [21] only guarantee a stretch of $\geq \Delta$, and techniques in [20] 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 [4], 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 $P_{ans}$ 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 $\epsilon \cdot |P_{ans}|$.

High-degree hierarchy: A first attempt. Given the “high-degree” obstacle, it is natural to see whether the “high-degree hierarchy” of [31] may help us. Plugging the spanners\(^5\) into the hierarchy of [31], 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 $i$. 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 $P_{ans}$ 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 [31, 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

\(^5\)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 $U_i$. 

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited
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 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, 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 3.2. 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 $O(1)$ 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 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 $G - (X \cup U_i)$ 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 4.1) for shortest paths under vertex failures, which allows us to compress such paths. (The corresponding theorem [20, Theorem 3.1] does not hold for vertex failures.) Curiously, the proof of this theorem also relies on Lemma 3.2.

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 poly$(\log n)$ query time, within a reasonable stretch.\footnote{It seems that even $O(\sqrt{n})$ stretch was open before this result.}

Note that setting $\epsilon = \omega(1)$ (e.g. $\epsilon = \log n$) in oracle (a) does not improve its space complexity to $n^2 \log^{d(\epsilon)n}$, so oracle (b) is not a direct corollary of oracle (a).

1.4 Notation. In this paper, $\log x = \log_2 x$, $\ln x = \log_e x$. For a set $S$ and an integer $k$, $|S|$ is the cardinality of $S$, and we denote $|S|^k = \{ S' \subseteq S : |S'| = k \}$, and $|S|_{\leq k}, |S|_{\geq k}$ 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 $G$ 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 [25]).

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 $P(u, v) = [x_1, v], P(u, v) = P[u, x_{\ell-1}]$ and $P(u, v) = [x_1, x_{\ell-1}]$. 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 $\text{dep}_T(x)$, as the (weighted) distance from $x$ to $r$ in $T$.

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

Note that we also define some more notations at the end of Section 2.2, which is relevant to the “high-degree hierarchy”.

2 Source-Restricted Tree Covers in High-Degree Hierarchy

Our VSDO is based on a variant of the high-degree hierarchy of [31], which we equip with the source-restricted...
tree covers of [60, 53] 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 [53], we call such tree covers source-restricted.

Throughout this paper, \( k = \ln n \). 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 the distance between \( u \) and \( v \) in \( T(w) \) is at most \( 2k-1)\delta(u, v) \).

c) Every vertex \( v \in V \) is in at most \( kn^{1/k}(\ln n + 1) \leq 2e\ln^2 n \) trees.

In [60], 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 [53]) yields source-restricted tree covers.

**Theorem 2.1.** 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.

A proof sketch of Theorem 2.1 can be found in Appendix B of the full version; we also refer the interested reader to [60, 53] for details.

For \( S \subseteq V \), we denote \( \mathcal{T}(S) \) as the \( S \)-restricted tree cover constructed in Theorem 2.1. 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 \( V \).

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 2.2.** 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 \( u, v \in V(T(w)) \), and \( v \) lies on the path from \( u \) to \( w \) in \( T \). The subtree (subgraph) of \( T \) induced by trunk vertices of \( T \) is denoted as \( \text{Trunk}(T) \). The pseudo-degree of a vertex \( v \in V(T) \), denoted as \( \text{pdeg}_T(v) \), is the degree of \( v \) in \( \text{Trunk}(T) \). If \( v \) is not a trunk vertex of \( T \), then \( \text{pdeg}_T(v) = 0 \).

Note that vertices in \( \text{Trunk}(T) \) 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 \( \text{Trunk}(T) \), its path in \( T \) to any vertex in \( S \) must go through its parent. (This is because the root of \( T \) is always in \( S \).)

Let \( s = 4e \cdot d^c \ln^2 n + 1 \) be a degree threshold, where \( c \geq 1 \) is any constant. Define \( \text{Hi}(\mathcal{T}(S)) \) 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 \( |\text{Hi}(\mathcal{T}(S))| \).

**Lemma 2.1.** For any \( S \subseteq V \), \( |\text{Hi}(\mathcal{T}(S))| \leq \frac{|S|}{2e^{d+c}} \).

**Proof.** For a tree \( T \), let \( \text{Leaf}(T) \) be the set of leaves of \( T \). Then there are at most \( \frac{|\text{Leaf}(T)|-2}{s-1} \) vertices in \( T \) that has degree \( > s \) [31, Lemma 3.1]. For any \( T \in \mathcal{T}(S) \), \( \text{Leaf}(\text{Trunk}(T)) \subseteq S \) by definition, thus

\[
\sum_{T \in \mathcal{T}(S)} |\text{Leaf}(\text{Trunk}(T))| \leq \sum_{v \in S} |\{T \in \mathcal{T}(S) : v \in T\}|.
\]

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.

<table>
<thead>
<tr>
<th></th>
<th>degree</th>
<th>pseudo-degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r )</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>( u )</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>( v )</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>( w )</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Copyright © 2021 by SIAM
Unformatted reprint of this article is prohibited
We use a simplified version of the high-degree hierarchy in [31]. 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 2.3.** The hierarchy tree is a rooted tree in which every node\(^{10}\) 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 $G \setminus W$. The hierarchy tree is constructed as follows. Let $U$ be any node. If $\text{Hi}(\mathcal{T}(U)) = \emptyset$, then $U$ is a leaf; otherwise let $W_1, W_2, \ldots, W_d$ be its children, where

\begin{equation}
W_1 = \text{Hi}(\mathcal{T}(U)),
W_i = W_{i-1} \cup \text{Hi}(\mathcal{T}_{W_{i-1}}(U)). \quad \text{(for } 2 \leq i \leq d)\end{equation}

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

There are two main differences compared with the hierarchy structure in [31].

\begin{itemize}
  \item We simplified the definition of $W_1$ as in (2.1). 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 [31] (and Section 1.3).
  \item More importantly, we store in each node the source-restricted tree covers introduced in Section 2.1. By contrast, [31] only concerns about connectivity, so they used a (somewhat arbitrary) spanning forest instead.
\end{itemize}

The following lemmas assert that the hierarchy tree “is small, shallow and effectively represents the graph” [31], which are crucial for our data structures.

**Lemma 2.2.** (Hierarchy Size and Depth)
Consider the hierarchy tree constructed with high-degree threshold $s = 4c \cdot d^{c+1} \ln^2 n + 1$, then the following hold.

1. The depth $h$ of the hierarchy tree is at most $\lfloor \frac{1}{2} \log_d n \rfloor$,\(^{11}\) assuming the root has depth 0.
2. The number of nodes in the hierarchy tree is at most $O(n^{1/c})$.

\begin{proof}
Let $U$ be a node in the hierarchy tree, $W_1, W_2, \ldots, W_d$ be its children (if exist). By Lemma 2.1, we have $|W_i| = |\text{Hi}(\mathcal{T}(U))| \leq \frac{|U|}{d^{i-1}}$ and $|\text{Hi}(\mathcal{T}_W(U))| \leq \frac{|U|}{d^{i-1}}$ for any $0 < i \leq d$. Since $W_{i+1} = W_i \cup \text{Hi}(\mathcal{T}_{W_i}(U))$, it follows that $|W_i| \leq i|U|/2d^{i+1}$ for each $i$. Therefore $|W_i| \leq |U|/2d^i$ for all $0 < i \leq d$. Any node at the $k$-th level corresponds to a subset of $V$ with size at most $n/(2d^k)^k$, therefore the depth of the hierarchy tree is at most $h \leq \lfloor \log_{2d} n \rfloor \leq \lfloor \frac{1}{2} \log_d n \rfloor$. There are at most $\sum_{i=0}^{h} d^i \leq 2dh = O(n^{1/c})$ nodes in the hierarchy tree.
\end{proof}
Lemma 2.3. 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 \( T \in \bigcup_{i \leq p} T_i \) and every \( f \in D, f \) has pseudo-degree \( \leq s \) in \( T \). (Assume \( U_{p+1} = \emptyset \).)

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 \( W_1 = \text{Hi}(T(U_p)) \) is the first child of \( U_p \). Clearly, in both cases we have \( D \cap \text{Hi}(T(U_p)) = \emptyset \).

For any \( 1 \leq i < p \), 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 \( D \cap (W_j \setminus W_{j'}) = \emptyset \) for \( j' = 0, \ldots, d - 1 \). Since \( |D| \leq d \), we have \( D \subseteq W_d \), thus \( D \cap \text{Hi}(T(W_i)) = \emptyset \). If \( 1 \leq j < d \), then we have \( D \cap (W_j \setminus W_j) = \emptyset \). Since \( W_{j+1} = W_j \cup \text{Hi}(T(W_i)) \) and \( \text{Hi}(T(W_i)) \cap W = \emptyset \), we have \( D \cap \text{Hi}(T(W_i)) = \emptyset \). 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 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 2.3, then every failed vertex \( f \in D \) has pseudo-degree \( \leq s \) in every \( T_i \) \( (1 \leq i \leq p) \).

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 2.1 will be important for us.

Corollary 2.1. Let \( x \in U_i \setminus U_{i+1} \) and \( y \in V \setminus U_{i+1} \). There is a tree \( T \in \bigcup_{i \leq i+1} T_i \) such that \( x, y \in T \), and \( T \) is at most \( (2k-1)\delta_{G-U_{i+1}} \). Moreover, the root of such a tree can be found in \( O(k) \) time.

We will denote this tree as \( T_x(y, x) \), and denote the path from \( x \) to \( y \) in \( T \) as \( P_{x}(y, x) \).

The set of trees is denoted as \( T = \bigcup_{i \leq i+1} T_i \). The proof of Lemma 2.2 shows that \( |U_i| = |U_{i-1}| = 2d^p \), therefore \( |T| \leq \sum_{i \leq i+1} |U_i| = O(n) \). 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 2.4. 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 \( G_l \) to be the subgraph of \( G \) induced by all vertices with level at most \( l \).

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

Space

In this section we present an oracle with

- space complexity \( n^{3+1/c} \cdot (\epsilon^{-1} \log(nW))^{O(d)} \),
- query complexity \( \text{poly}(\log(nW), \epsilon^{-1}, d) \),
- stretch \( 1 + \epsilon \),

for any \( \epsilon > 0 \). In this paper (except Section 5 and Section 4.5) we may assume \( d = o\left(\frac{\log n}{\log \log n}\right) \), \( W = \text{poly}(n) \), 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.

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 [20, 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 \( P = (u = v_0, v_1, v_2, \ldots, v_l = v) \). For every \( 1 \leq i, j \leq l \), we say \( v_i \) and \( v_j \) are in the same segment if one of the two conditions hold:

- \( |P[u,v_i]|, |P[u,v_j]| \leq |P|/2 \) and \( |\log_{1+\epsilon_2} |P[u,v_i]| | < |\log_{1+\epsilon_2} |P[u,v_j]| | \).
- \( |P[v_i,v]|, |P[v_j,v]| < |P|/2 \) and \( |\log_{1+\epsilon_2} |P[v_i,v]| | < |\log_{1+\epsilon_2} |P[v_j,v]| | \).

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 \( \text{seg}(v_i, P) \) as the segment it belongs to. More precisely, \( \text{seg}(v_i, P) = P[v_i, v_r] \) where \( v_r \) is the leftmost (closest-to-\( u \)) vertex in the segment, and \( v_r \) is the rightmost (closest-to-\( v \)) vertex in the segment. Define \( \text{seg}(P) \) as the set of segments on \( P \). Note that \( u \) and \( v \) do not belong to any segment.

Lemma 3.1. Let \( P \) be a path from \( u \) to \( v \), \( x \in P \setminus \{u,v\} \), then \( |\text{seg}(x,P)| \leq \epsilon_2 \cdot \min(|P[u,x]|,|P[x,v]|) \).

Unauthorized reproduction of this article is prohibited.
Proof. If $|P[u, x]| \leq |P|/2$, $|P[u, x]| \leq |P[x, v]|$ and $\operatorname{seg}(x, P)$ is defined in the first way. Let $i = \lceil \log_{1+\epsilon_2} |P[u, x]| \rceil$. For any $y \in \operatorname{seg}(x, P)$, $(1 + \epsilon_2)^i \leq |P[u, y]| < (1 + \epsilon_2)^{i+1}$. Thus $|\operatorname{seg}(x, P)| \leq (1 + \epsilon_2)^{i+1} - (1 + \epsilon_2)^i = \epsilon_2(1 + \epsilon_2)^i \leq \epsilon_2|P[u, x]|$. The case that $|P[x, v]| < |P|/2$ is symmetric. \hfill \QED

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\textsuperscript{12} \(\alpha \in FT(u, v)\) is associated with a set $\text{avoid}(\alpha) \subseteq V$ of vertices that we avoid.
- Denote the root of $FT(u, v)$ as $\text{root}(u, v)$, and let $\text{avoid}(\text{root}(u, v)) = \emptyset$.
- For each node $\alpha \in FT(u, v)$, we store the path $P_\alpha = \pi_{G-\text{avoid}(\alpha)}(u, v)$, i.e. the shortest $u$-$v$ path in $G$ not passing through avoid(\alpha). If $u, v$ are not connected in $G$ - 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 $X \in \text{seg}(P_\alpha)$, and each $1 \leq i \leq p$, we create a child $ch(\alpha, X, i)$ of $\alpha$, in which
  $\text{avoid}(ch(\alpha, X, i)) = \text{avoid}(\alpha) \cup (X \cap U_i)$.

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

The decision tree has depth $d$, and each non-leaf node has $O(p \cdot \log |P|/\epsilon_2) = O(hc^{-1} \log n \log(nW))$ 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 $\text{seg}(P_\alpha)$ for each $v \in P_\alpha$, so that we can quickly locate any vertex in $P_\alpha$. Therefore one decision tree occupies $n \cdot O(hc^{-1} \log n \log(nW))^d$ 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 $n^{3+1/c} \cdot O(hc^{-1} \log n \log(nW))^d$.

\textsuperscript{12}Recall that we use “vertex” for nodes in the input graph, and “node” for nodes in the decision trees and the hierarchy tree.

\textsuperscript{13}The vertex set of $H$ corresponds to the set of “intermediate vertices” (also called $H$) in Section 1.3.

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

3.2 Query algorithm. Given $u, v$ and a set of failed vertices $D$, by Lemma 2.3 we first find a path $U_1(= V), U_2, \ldots, U_p$ in the hierarchy tree, and set $U_{p+1} = \emptyset$. Let $T = \bigcup_{i=1}^p \mathcal{T}_{U_{i+1}}$ $$(U_i)$$, then by Lemma 2.3, the pseudo-degrees of all $f \in D$ in every tree in $T$ is at most $s$.

As in [20], the query algorithm builds an auxiliary graph $H$, but the definition of $H$ is different from [20]. The query algorithm builds $H$ as Definition 3.2, and outputs $\tau_H(u, v)$ as an $(1 + \epsilon)$-approximation of $\pi_{G-D}(u, v)$.

**Definition 3.2.** (Graph $H$)

- For a failure $f \in D$ and a tree $T \in T$, if $f \in V(T)$, then we define the neighbors of $f$ in $T$ as
  $N_T(f) = \{\text{parent}_T(f)\} \cup (\text{children}_T(f) \cap \text{Trunk}(T))$.

  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 $f \in V(T) \setminus \text{Trunk}(T)$, then it is possible that parent$_T(f) \notin \text{Trunk}(T)$.)

- Define
  $$N(f) = \bigcup_{f \in T} N_T(f).$$

That is, $N(f)$ is the union of $N_T(f)$’s over all trees $T \in T$ 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 $\text{DecTree}(x, y, D)$, as defined in Algorithm 2.

Note that in $V(H)$, vertices except $u$ and $v$ are defined independently from $u$ and $v$. By Lemma 2.3, for every $f \in D$ and $T \in T$ containing $f$, we have $|N_T(f)| \leq s + 1$. Therefore $|V(H)| \leq dp \cdot 2 \epsilon \ln^2 n \cdot (s + 1) = O(d^{r+2}h \log^4 n)$. 

Copyright © 2021 by SIAM 
Unauthorized reproduction of this article is prohibited
Algorithm 2 Algorithm DecTree

1: function DecTree(u, v, D)
2:    \( \alpha \leftarrow \text{root}(u, v) \)
3:    \( \text{while } D \cap \alpha \neq \emptyset \text{ do} \)
4:        \( f \leftarrow \text{the vertex in } D \cap \alpha \text{ with the highest level, breaking ties arbitrarily} \)
5:        \( \alpha \leftarrow \text{ch}(\alpha, \text{seg}(f, P_\alpha), l(f)) \) \( \triangleright \) Recall \( l(f) \) is the level of \( f \), i.e., the largest \( l \) s.t. \( f \in U_l \).
6:    return \(|P_\alpha|\)

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 \( D \cap \alpha = \emptyset \), 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 \( O(|V(H)|^2 d^2) = O(d^2 + h^2 \log^8 n) \) time.

Since finally \( D \cap \alpha = \emptyset \), we have the following observation:

Observation 3.1. For all \( u, v \in V(H), \) DecTree\((u, v, D) \geq |\pi_{G-D}(u, v)|\).

For every \( \alpha \) and \( f \) considered in Algorithm 2, let \( \alpha_{\text{next}} = \text{ch}(\alpha, \text{seg}(f, P_\alpha), l(f)) \) 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 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 3.2. 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 \( T_{U_{j+1}}(U_j) \), 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 \( P' = P_j(y, f) \) be the path between \( y \) and \( f \) in \( T_j(y, f) \). Let \( P' \) 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.

1. If \( w \) is the parent of \( f' \), then \( w \in V(H) \) by the definition of \( V(H) \).
2. 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

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

the lemma is true. \( \square \)

3.3 Proof of correctness. In this section, we show that \(|\pi_H(u, v)| \leq (1 + \epsilon)|\pi_{G-D}(u, v)|\); 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 \( \text{avoid}(\alpha_{\text{last}}) \), then DecTree\((u, v, D) \) will return the correct answer. Otherwise, as \( \text{avoid}(\alpha_{\text{last}}) \) is the union of \( d \) 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 3.2), so we can use the vertices in \( V(H) \) as intermediate vertices to obtain an approximate shortest path.

Lemma 3.3. In the query algorithm DecTree\((u, v, D) \), let \( \alpha \) be a decision tree node if encountered, \( f \in D \) 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 \(|\pi_{G-D}(x, w)| \leq \epsilon_1 \min\{|P_\alpha[u, x]|, |P_\alpha[x, v]|\}| \).
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 \( l(f) \geq l(f') \) by Algorithm 2. As \( x \in U_1 = U_\ell(f) \), we have \( l(x) \geq l(f) \), hence \( l(x) \geq l(f') \).

By Lemma 3.2, there is a vertex \( w \in V(H) \) such that \( |\pi_{G-D}(x, w)| \leq (2k-1)|P_\alpha[x, f']| \). Since \( |P_\alpha[x, f']| \leq \epsilon_2 \min\{|P_\alpha[u, x]|, |P_\alpha[x, v]|\} \) 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 \( \text{DecTree}(u, v, D) \), 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 3.4. Let \( u, v \in V(H) \) and \( P = \pi_{G-D}(u, v) \). If \( \text{DecTree}(u, v, D) > |P| \), there exist \( x \in P \setminus \{u, v\}, y \in \{u, v\}, w \in V(H) \) such that

(a) \( |\pi_{G-D}(x, y)| \leq \frac{1}{2}|\pi_{G-D}(u, v)| \),

(b) \( |\pi_{G-D}(x, w)| \leq \epsilon_1|\pi_{G-D}(x, y)| \), and

(c) \( \text{DecTree}(y, w, D) \leq |\pi_{G-D}(y, x)| + |\pi_{G-D}(x, w)| \),

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 \( \text{DecTree}(u, v, D) \) such that \( \text{avoid}(\alpha) \cap P = \emptyset \). Since \( \text{DecTree}(u, v, D) > |P| \), the procedure \( \text{DecTree}(u, v, D) \) did not terminate at \( \alpha \), i.e. it visited a child \( \alpha_{\text{next}} \) of \( \alpha \) such that \( P \cap \text{avoid}(\alpha_{\text{next}}) \neq \emptyset \). Recall that \( \text{avoid}(\alpha_{\text{next}}) \setminus \text{avoid}(\alpha) = \text{seg}(f, P_\alpha) \cap U_\ell(f) \) where \( f \) is the failure selected by Line 4 of Algorithm 2. Therefore \( P \) reaches some vertex \( x \in \text{seg}(f, P_\alpha) \cap U_\ell(f) \). Since \( P_\alpha \) is the shortest \( u-v \) path in \( G \)-avoid(\( \alpha \)) and \( P \) is some \( u-v \) path in \( G \)-avoid(\( \alpha \)), we know that \( |P_\alpha[u, x]| \leq |P[u, x]| \) and \( |P_\alpha[x, v]| \leq |P[x, v]| \).

By Lemma 3.3, there is a vertex \( w \in V(H) \) such that

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

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 \( \text{DecTree}(y, w, D) \). For the sake of contradiction, suppose \( |P'| < \text{DecTree}(y, w, D) \). Let \( \alpha' \) be the last decision tree node visited in \( \text{DecTree}(y, w, D) \) such that \( \text{avoid}(\alpha') \cap P' = \emptyset \). We can also see that \( P' \) reaches some vertex \( x' \in \text{seg}(f', P_{\alpha'}) \cap U_\ell(f') \), where \( f' \) is the failure selected by Line 4 of Algorithm 2. We use Lemma 3.3 again and conclude that there is a vertex \( w' \in V(H) \) such that

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

Since \( x' \in P' = \pi_{G-D}(y, x) \circ P'[x', w] \), there are two cases. (See Fig. 3.)

- If \( x' \in P'[y, x] \), then \( |\pi_{G-D}(x', w')| \leq \epsilon_1|\pi_{G-D}(x', y)| \), 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 \( |\pi_{G-D}(x', w')| \leq |P'[x, x']| + |\pi_{G-D}(x', w')| \leq |P'[x, w']| + \epsilon_1 |P'[x', w]| \) 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 \( |P'| \geq \text{DecTree}(y, w, D) \).

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

Theorem 3.1. 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 \(|\pi_H(u, v)| \geq |\pi_{G-D}(u, v)| \) for every \( u, v \in V(H) \). We prove \(|\pi_H(u, v)| \leq (1 + \epsilon)|\pi_{G-D}(u, v)| \) below.
We sort all pairs of vertices \(u, v \in V(H)\) (\(u \neq v\)) by increasing order of \(|\pi_{G-D}(u, v)|\), and prove by induction that \(|\pi_H(u, v)| \leq (1 + \epsilon)|\pi_{G-D}(u, v)|\) 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 3.4, there exist \(y, w \in V(H)\) so that \(|\pi_{G-D}(y, w)| < |\pi_{G-D}(u, v)|\), which is a contradiction. Therefore \(|\pi_H(u, v)| = \text{DecTree}(u, v, D) = |\pi_{G-D}(u, v)|\).

Fix some \(u, v \in V(H)\), assume for all pairs \(u', v' \in V(H)\) such that \(|\pi_{G-D}(u', v')| < |\pi_{G-D}(u, v)|\), it is true that \(|\pi_H(u', v')| \leq (1 + \epsilon)|\pi_{G-D}(u', v')|\). If DecTree\((u, v, D) = |\pi_{G-D}(u, v)|\) then \(|\pi_H(u, v)| \leq (1 + \epsilon)|\pi_{G-D}(u, v)|\) follows trivially. Otherwise we use Lemma 3.4 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 \(|\pi_{G-D}(w, x)| \leq \epsilon_1|\pi_{G-D}(u, x)|\). Since \(\epsilon_1 < 1\), \(|\pi_{G-D}(w, x)| \leq |\pi_{G-D}(x, v)| + |\pi_{G-D}(x, v)| \leq \epsilon_1|\pi_{G-D}(u, x)| + |\pi_{G-D}(x, v)|\), thus \(|\pi_H(u, v)| \leq (1 + \epsilon)|\pi_{G-D}(w, v)|\) by induction hypothesis. We have:

\[
|\pi_H(u, v)| \\
\leq \text{DecTree}(u, w, D) + |\pi_H(w, v)| \\
\leq |\pi_{G-D}(u, x)| + |\pi_{G-D}(w, v)| + (1 + \epsilon)|\pi_{G-D}(w, v)| \\
\leq |\pi_{G-D}(u, x)| + \epsilon_1|\pi_{G-D}(u, x)| + (1 + \epsilon)|\pi_{G-D}(w, v)| \\
\leq (1 + \epsilon_1 + (1 + \epsilon)|\pi_{G-D}(u, x)| + (1 + \epsilon)|\pi_{G-D}(w, v)| \\
\leq (1 + \epsilon)|\pi_{G-D}(u, v)|.
\]

We conclude that there is a VSDO with

space complexity \(n^{3+1/c} \cdot O\left(\frac{1}{d} \cdot \log^2 n \cdot \log(nW)\right)^d\),

query complexity \(O\left(\frac{d^{2+\epsilon} \cdot \log^2 n}{\log d}\right)\),

stretch \(1 + \epsilon\).

In Section 4, we will improve the space complexity to \(n^{2+1/c+o(1)}\) when \(d = o\left(\frac{\log n}{\log \log n}\right)\) and \(W = \text{poly}(n)\), 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 3.2 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 [20], the main idea is, instead of storing the paths \(P_a\) as-is in every node \(\alpha\) of \(FT(u, v)\), we store an implicit representation of these paths. If the representation has size \(poly(\log(nW)\), \(\epsilon^{-1}\)) instead of \(\Omega(n)\), then our data structure has space complexity \(n^{2+1/c+o(1)}\).

In [20], 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 [20] 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 \(k = o(n)^{14}\).

In this section, we prove a structural theorem similar to the above fact used in [20]. Before we proceed, we need some definitions.

From Lemma 3.4 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 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 \(|\pi_{G-D}(x, w)| \leq \epsilon \cdot \min\{|P[u, x]|, |P[x, v]|\}\). (See Fig. 4.)

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 [20] 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

---

Figure 3: Two cases in the proof of Lemma 3.4.
4.2 New data structure. We generalize the concept of $\epsilon_4$-segment expath to $\epsilon_t$-expath by adding more flexibility.

**Definition 4.3.** Let $B = \lceil \log_{1+\epsilon_4}(nW) \rceil$. An $\epsilon_t$-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\(^\text{15}\), such that the following hold.

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

**Lemma 4.1.** An $\epsilon_t$-segment expath $P$ from $u$ to $v$ is an $\epsilon_t$-expath.

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

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 $\text{avoid}(\alpha)$ 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 3.2, 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 $1 \leq i \leq p$, 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 $\epsilon_t$-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

\footnote{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.}
shortest paths (see [28]), 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 3.3 holds for parameter $(2k-1)\epsilon_i = \epsilon_3/2$, as follows.

**Remark of Lemma 3.3.** In the query algorithm $\text{DecTree}(u,v,D)$, 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 $|\pi_{G-D}(x,w)| \leq (\epsilon_3/2) \min\{|P_\alpha[u,x]|,|P_\alpha[x,v]|\}$. Recall that Lemma 3.4 shows that, in the data structure in Section 3, any shortest path $c_1$-far away from $V(H)$ can be found by $\text{DecTree}$. We show that this is also true in the new data structure, where “$c_1$-far away” is changed to “$c_3$-far away”.

**Lemma 4.2.** Let $u,v \in V(H)$, and $P = \pi_{G-D}(u,v)$. If $\text{DecTree}(u,v,D) \neq |P|$, then $P$ is not $c_3$-far away from $V(H)$.

**Proof.** For the sake of contradiction, assume $P$ is $c_3$-far away from $V(H)$. By Theorem 4.1, $P$ is an $\epsilon_4$-segment expath.

Let $\alpha$ be the last decision tree node visited by $\text{DecTree}(u,v,D)$ such that $|P_{\alpha} \cap U_i| \neq 0$. Since $\text{DecTree}(u,v,D) \neq |P|$, $P$ 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 $\text{avoid}(\alpha_{\text{next}}) \setminus \text{avoid}(\alpha) = \text{seg}(f,P_\alpha) \cap U_i(f)$, where $f$ is the failed vertex chosen in Line 4 of Algorithm 2.

By Lemma 3.3, there is a vertex $w \in V(H)$ such that

$$|\pi_{G-D}(x,w)| \leq (\epsilon_3/2) \min\{|P_\alpha[u,x]|,|P_\alpha[x,v]|\}.$$

As $P_{\alpha}$ is the shortest $c_1$-expath from $u$ to $v$ in $G \setminus \text{avoid}(\alpha)$, and $P$ is some such path, we have $|P| \geq |P_{\alpha}|$. We will prove $|P_{\alpha}[u,x]| \leq 2|P[u,x]|$ and $|P_{\alpha}[x,v]| \leq 2|P[u,x]|$, then it will follow that $|\pi_{G-D}(x,w)| \leq \epsilon_3 \min\{|P_{\alpha}[u,x]|,|P_{\alpha}[x,v]|\}$, contradicting that $P$ is $c_3$-far away from $V(H)$. We only prove $|P_{\alpha}[u,x]| \leq 2|P[u,x]|$, and the case that $|P_{\alpha}[x,v]| \leq 2|P[u,x]|$ is symmetric.

Suppose $|P[u,x]| < |P_{\alpha}[u,x]|/2$, we claim that the path $P_{\alpha}[u,x] \circ P_{\alpha}[x,v]$ is a valid $\epsilon_4$-expath. Since $|P[u,x]| < |P_{\alpha}[u,x]|/2$, $x$ 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 $\leq 2B + 3$ edges, and $P$ is composed of segments $P_1, \ldots, P_k$. (Every $P_{\alpha}^i$ and $P_j$ is a shortest path in some $G_i$.) Recall from the proof of Lemma 4.1 that, if $x$ is in the first half of $P$, and $x \in P_k$, then $|\log_{1+\epsilon_4}|P_{\alpha}[u,x]|| = k - 1$.

- Let $x \in P_{\alpha}^j$, then $j \geq |\log_{1+\epsilon_4}|P_{\alpha}[u,x]||$. This is because if $j < B+1$ (recall that $B = |\log_{1+\epsilon_4}(nV)|$) as in Definition 4.3), then $|P_{\alpha}[u,x]| \leq (1 + \epsilon_4)^j$.
- Let $x \in P_{\alpha}^j$, then $j' = [\log_{1+\epsilon_4}|P_{\alpha}[u,x]|| + j - 1$. Since $(1 + \epsilon_4)^2 < 2 \leq |P_{\alpha}[u,x]|/|P[u,x]|$, we have $j' \leq |\log_{1+\epsilon_4}|P_{\alpha}[u,x]|| - 1 \leq j - 1$.

Let $P' = P[u,x] \circ P_{\alpha}[u,x,v]$. Consider the following representation of $P'$ as $P_{\alpha}^0, P_{\alpha}^1, \ldots, P_{\alpha}^{2B+1}$:

(i) For $0 \leq i < j'$, $P_i' = P_i$.
(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 $i = j$, $P_j' = P_j^0[x,v_j^0]$, where $v_j^0$ is the endpoint of $P_j^0$ that lies on $P_{\alpha}[x,v]$.
(v) For $j < i \leq 2B+1$, $P_i' = P_i^\alpha$.

We need to verify that the representation $P_{\alpha}^0, P_{\alpha}^1, \ldots, P_{\alpha}^{2B+1}$ satisfies the definition of $\epsilon_4$-expath. Let $u_i', v_i'$ be the endpoints of $P_i'$, i.e. $P_i' = P_{\alpha}^i[u_i', v_i']$, then:

- Case I: $i \leq j'$ (i.e. Items i and ii). In this case, $i < B+1$, and $P_i'$ lies in the first half of $P$. Since $|P_{\alpha}^i[u_i', v_i']| = |P_i[u_i', v_i']| \leq (1 + \epsilon_4)^i$, Definition 4.3 is satisfied.
- Case II: $j \leq i < B+1$. In this case, $|P_{\alpha}^i[u_i', v_i']| = |P_{\alpha}^i[u_i', v_i']| + |P[x,v]| < |P_{\alpha}^i[u_i', v_i']| \leq (1 + \epsilon_4)^i$, thus Definition 4.3 is satisfied.
- Case III: $j \geq B+1$. In this case, $|P_{\alpha}^i[u_i', v_i']| = |P_{\alpha}^i[u_i', v_i']| \leq (1 + \epsilon_4)^{2B+1-1}$, thus Definition 4.3 is satisfied.
We conclude that $P'$ is a valid $\epsilon_4$-expath. Since $|P'| < |P_a|$, this contradicts the choice of $P_a$.

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

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

**Theorem 4.2.** For every pair $u, v \in V(H)$, $|\pi_H(u, v)| \leq (1 + \epsilon)|\pi_{G-D}(u, v)|$.

**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 4.2, 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 $|\pi_{G-D}(x, w)| \leq \epsilon_3 \min\{|\pi_{G-D}(u, x)|, |\pi_{G-D}(x, v)|\}$. 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 $u, v \in V(H)$,

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

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 $|\pi_{G-D}(x, w)| \leq (\epsilon_3/2)|\pi_{G-D}(u, v)|$, we have

$$|\pi_H(u, v)| \leq |\pi_H(u, w)| + |\pi_H(w, v)| \leq (1 + \epsilon_3)^{j-1}(|\pi_{G-D}(u, w)| + |\pi_{G-D}(w, v)|) \leq (1 + \epsilon_3)^{j-1}(|\pi_{G-D}(u, v)| + 2|\pi_{G-D}(x, w)|) \leq (1 + \epsilon_3)^j(|\pi_{G-D}(u, v)| + 2\epsilon_3^j/2|\pi_{G-D}(u, v)|) \leq (1 + \epsilon_3)^j|\pi_{G-D}(u, v)|.$$

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

$$|\pi_H(u, v)| \leq |\pi_H(u, v)| \leq \left(1 + \frac{\epsilon}{2|V(H)|}\right)^{|V(H)|}|\pi_{G-D}(u, v)| \leq \epsilon^2|\pi_{G-D}(u, v)| \leq (1 + \epsilon)|\pi_{G-D}(u, v)|.$$

$\square$

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

As $\epsilon_4^{-1} = O(|V(H)| \cdot \epsilon^{-1} \log n) = O(d^2 \epsilon^{-1} h \log^5 n)$, the VSDO is of space complexity $n^{2+1/c} \cdot (\epsilon^{-1} d^{-1} \log(nW))^{O(d)}$, query complexity $O(\epsilon^{-1} d^{-1} \log W)$, stretch $1 + \epsilon$.

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 $\epsilon_4$ factor may seem too large. In this section, we show that the $|V(H)|$ factor in $\epsilon_4^{-1}$ can be shaven.

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 $a$, but we face a technical problem as follows. Suppose $\text{DecTree}(u, v, D)$ does not capture the shortest path $P = \pi_{G-D}(u, v)$, then by Lemma 3.4, $P$ is not far from $V(H)$. In other words, there are vertices $x \in P$ and $w \in V(H)$ such that $\pi_{G-D}(x, w) \leq \epsilon_1|P[u, x]|$. (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 3.4 shows that $P_1$ is far away from $V(H)$, so we may attempt to use Lemma 4.2 to conclude that $|\text{DecTree}(u, w, D)| \leq |P_1|$, and we only need to “recurse” on $P_2$. However, Lemma 4.2 relies on Theorem 4.1, 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.
Theorem 4.3. For every path $P = \pi_{G-D}(u,v)$ in $G - D$, the concatenation of at most two shortest paths in $G_1$ and $G_2$. The original proof of Theorem 4.1 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 4.4. 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 $P_{\pi}[u,v]$, there exist two levels $i,j$ and a vertex $z \in P_{\pi}[u,v]$ such that $P_{\pi}[u,v] = \pi_G(u,v,z) \circ \pi_{G_j}(z,v_k)$.

The following theorem can be proved by similar arguments as Theorem 4.1.

Theorem 4.3. 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 4.1 applies to $P[x,y]$, and there is some $1 \leq i \leq p$ such that $P[x,y] = \pi_G(x,y)$. 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 4.1, and we can see that $P[x,v]$ is the shortest $x$-$v$ path in $G_{i(z_1)}$, and $P[v,y]$ is the shortest $v$-$y$ path in $G_{i(z_2)}$.

Similarly we can define $\epsilon_5$-bipaths:

Definition 4.5. Let $B = \lceil \log_2(1+\epsilon_5(nW)) \rceil$. An $\epsilon_5$-bipath $P$ from $u$ to $v$ in $G$ is a path which is a concatenation of subpaths $P_0, \ldots, P_{2^B+1}$ interleaved with at most $2B + 3$ edges, such that the following hold:

- For every $P_k = P[u_k,v_k] \ (0 \leq k \leq 2B+1)$, either $P_k$ is empty, or there exists a vertex $z \in P[u_k,v_k]$ and two levels $1 \leq i,j \leq p$, such that $P[u_k,v_k] = \pi_G(u_k,z) \circ \pi_{G_j}(z,v_k)$.

- If $k < B+1$, then $|P[u_k,v_k]| \leq (1+\epsilon_5)^k$; if $k \geq B+1$, then $|P[u_k,v_k]| \leq (1+\epsilon_5)^{2B+1-k}$.

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 3.3 still holds (for parameter $(2k-1)\epsilon_5 = \epsilon_1/2$).

Reminder of Lemma 3.3. 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 $i = \ell(f)$. (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 $|\pi_{G-D}(x,w)| \leq (\epsilon_1/2) \min \{|P_{\alpha}(u,x)|, |P_{\alpha}(x,v)|\}$.

It is easy to verify that the counterparts of Lemma 4.1 and Lemma 4.2 also hold for (segment) bipaths.

Lemma 4.3. An $\epsilon_5$-segment bipath $P$ from $u$ to $v$ is an $\epsilon_5$-bipath.

Lemma 4.4. (Assume $1 + \epsilon_5 < \sqrt{2}$.) Let $y,w \in V(H)$, $x \in V$, and $P = \pi_{G-D}(y,w,x) \circ \pi_{G-D}(x,w)$. If $\text{DecTree}(y,w,D) > |P|$, then $P$ is not $\epsilon_1$-far away from $V(H)$.

Proof. (Sketches of Lemma 4.3 and Lemma 4.4.) The arguments are essentially the same as Lemmas 4.1 and 4.2, except that each subpath in $P$ and $P_{\alpha}$ is now a concatenation of two shortest paths in $G_i$ and $G_j$. This does not affect the calculation of lengths of paths in the proofs. In particular, in Lemma 4.2, 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 4.4. For every $u,v \in V(H)$, $|\pi_H(u,v)| \leq (1 + \epsilon)|\pi_{G-D}(u,v)|$.

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 $|\pi_{G-D}(u,v)| = |\pi_{G-D}(u,v)|$ 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 $|P[x,y]| \leq \frac{1}{2}|P|$ and $|\pi_{G-D}(x,w)| \leq \epsilon_1|P[x,y]|$.

Among all such triples $(x,y,w)$, we choose the triple that minimizes $|P[x,y]|$, and in case of ties choose the
4.4 Implementation details.

If \( x' \in P'(u, x) \), then the triple \((x', y, w')\) also satisfies that \( |\pi_{G-D}(x', w')| \leq \epsilon_1 |P'[x', y]| \) 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 \( |\pi_{G-D}(x, w')| \leq |P'[x, x']| + |\pi_{G-D}(x, w')| \leq |P'[x, x']| + \epsilon_1 |P'[x', w']| < |\pi_{G-D}(x, w)| \) 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 4.4, we have DecTree\((u, w, D) \leq |P'| = |\pi_{G-D}(u, x)| + |\pi_{G-D}(x, w)| \). It is easy to see that \( |\pi_{G-D}(w, v)| < |\pi_{G-D}(u, v)| \), thus by induction hypothesis \( |\pi_{H}(v, w)| \leq (1 + \epsilon)|\pi_{G-D}(v, w)| \). We have

\[
|\pi_{H}(u, w)| \leq |\pi_{H}(u, w)| + |\pi_{H}(w, v)| \\
\leq |\pi_{G-D}(u, x)| + |\pi_{G-D}(x, w)| \\
+ (1 + \epsilon)(|\pi_{G-D}(w, v)|) \\
\leq |\pi_{G-D}(u, x)| + (1 + \epsilon + (1 + \epsilon)\epsilon_1) + (1 + \epsilon)|\pi_{G-D}(v, x)| \\
= (1 + \epsilon)|\pi_{G-D}(u, v)|.
\]

Since an \( \epsilon_5 \)-bipath occupies \( O(\epsilon^{-1}_5 \log(nW)) \) space, and each non-leaf node has \( O(h\epsilon^{-1}_5 \log(nW)) \) children, we have a VSDO of

\[
\text{space complexity} \quad n^{2+1/c\epsilon^{-1}_5 \log(nW)} \cdot O(h\epsilon^{-1}_5 \log(nW))^d, \\
\text{query complexity} \quad O(d^2V(H)^2 \cdot \epsilon^{-1}_5 \log(nW)),
\]

\[ \begin{aligned}
\text{stretch} & \quad 1 + \epsilon.
\end{aligned} \]

As \( \epsilon^{-1}_5 = O(\epsilon^{-1} \log n) \), the VSDO is of

\[
\text{space complexity} \quad n^{2+1/c\epsilon^{-1} \log(nW)} \cdot O(\epsilon^{-1}_5 \log^2 n \log(nW))^{d+1}, \\
\text{query complexity} \quad O\left(\frac{\epsilon^{-1}_5 \log^2 n \log(nW)}{\log^d d}\right),
\]

\[ \begin{aligned}
\text{stretch} & \quad 1 + \epsilon.
\end{aligned} \]

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 \( \min\{|\pi_{G}(s, t)| : 1 \leq p, \pi_{G}(s, t) \subseteq G'\} \) (which may be \( +\infty \)). 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 \( \leq j + 1 \) 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 closer-to-\( u \) one, 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 \( 0 \leq k \leq j \), \( P[s, v_k] \leq (1 + \epsilon')^k \).

We use a dynamic programming algorithm to compute \( |\pi(s, t, j)| \) for all \( k \leq B \). 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:

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

where

\[
(4.2) \quad |\pi(s, t, j)| = \min\{|\pi(s, u, j-1)| + w(u, v) + |\pi_{G'}(v, t)|\}.
\]

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 (4.2) 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 \( \pi_{G}(x, y) \). To check whether a failed vertex \( f \) is in an \( \epsilon' \)-expath \( P_{\alpha} \), we check every subpath \( \pi_{G}(x, y) \) whether it contains \( f \) by checking whether \( \pi_{G}(x, f) + \pi_{G}(f, y) = \pi_{G}(x, y) \). 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 \( |\log_{1+\epsilon'} |P_{\alpha}[u, f]| \) or \( |\log_{1+\epsilon'} |P_{\alpha}[f, v]| \).
If we store the distance matrices of each \( G_i \) during preprocessing, then every operation (i.e., checking if \( f \in \mathcal{P}_a \) and locating \( \text{seg}(f, \mathcal{P}_a) \)) can be done in \( O(e^{-1}\log(nW)) \) time. Therefore the time complexity of Algorithm 2 becomes \( O(e^{-1}\log(nW) \cdot d^2) \). Similar arguments also apply to \( e^\epsilon \)-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 \( 1 \leq i \leq p \), 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 \( 1 \leq i \leq p \) and \( u, v \in V(G_i) \), we can retrieve the path \( \pi_{G_i}(u, v) \) efficiently.

- Let \( \alpha \in \mathcal{FT}(u, v) \) be a decision tree node. Recall that \( \mathcal{P}_\alpha \) 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, \( \mathcal{P}_\alpha \) 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 3.2, and find the shortest \( u-v \) path in \( H \). Each edge \((x, y)\) in this path corresponds to a path returned by \( \text{DecTree}(x, y, D) \), which by Algorithm 2 is \( \mathcal{P}_\alpha \) for some decision tree node \( \alpha \). The concatenation of these paths \( \mathcal{P}_\alpha \) for each edge on \( \pi_H(u, v) \) forms an \((1+\epsilon)\)-approximate shortest \( u-v \) path in \( G-D \). As each \( \mathcal{P}_\alpha \) can be retrieved efficiently, this path can also be retrieved efficiently.

### 4.5 A reduction from arbitrary weights to bounded weights.

If \( W = n^{o(1)} \), 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 \( \log W \log^d \cdot n \log W \log^{d-1} n \) in the space complexity of our data structure, via a reduction from arbitrary weights to bounded weights. This reduction appears in [20, Lemma 4.1] and we notice that it also holds for vertex failures.

**Lemma 4.5. (Lemma 4.1 of [20], 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 \( O(Q \log W) \) query time and has stretch \((1 + 1/n)A\).

**Proof.** For every \( 0 \leq i \leq \log W / \log n \), 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 \( w \leq n^{i+1} \), then we have an edge \((u, v)\) of weight \([w \cdot n^{-(i+1)}]\) 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 \( \tilde{G}_i \). Then we use the oracle \( \mathcal{O}_i \) and \( \mathcal{O}_{i+1} \) to compute an \( A \)-approximation of the value

\[
\text{ans} = \min \{ \delta_{\tilde{G}_i}(u, v) \cdot n^{i-2}, \delta_{\tilde{G}_{i+1}}(u, v) \cdot n^{i-1} \}.
\]

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

We have \( \text{ans} \leq \delta_{G_{i_*}}(u, v) \cdot n^{i_*-2} \). For every edge \( e \in E(G) \) with weight \( w \), if \( e \) appears in the graph \( \tilde{G}_{i_*} \), then \( \left( [w \cdot n^{-(i_*-2)}] \cdot n^{i_*-2} \right) \leq w + n^{i_*-2} \), i.e. every such edge is “overestimated” by an additive error of at most \( n^{i_*-2} \). It follows that \( \text{ans} \leq \delta_{G_{i_*}}(u, v) + (n-1)n^{i_*-2} \). Since \( \delta_{G-D}(u, v) \geq n^{i_*} \), we have \( \text{ans} \leq (1 + 1/n)\delta_{G-D}(u, v) \).

As our new oracle computes an \( A \)-approximation of \( \text{ans} \), its stretch is \((1 + 1/n)A\). \( \square \)

Assuming \( \epsilon > 2/n \), Lemma 4.5 transforms the VSDO in Section 3 into a VSDO of

- space complexity \( n^{\frac{3 + \epsilon}{2}} \cdot \log W \log^d \cdot O\left(\frac{\log^3 n \log W}{\log^d} \right)^d \),
- query complexity \( O\left(\frac{\log^6 n \log^d W}{\log^d} \right) \),

and the VSDO in Section 4 into a VSDO of

- space complexity \( n^{\frac{2 + \epsilon}{2}} \cdot \log W \log^d \cdot O\left(\frac{\log^3 n \log W}{\log^d} \right)^d + 1 \),
- query complexity \( O\left(\frac{\log^6 n \log^d W}{\log^d} \right) \),

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

In the full version of this paper, we will also present a VSDO of

- space complexity \( n^{\frac{2 + \epsilon}{2}} \cdot \text{poly}(\log(nW), d) \),
- query complexity \( \text{poly}(\log(nW), d) \),
- stretch \( \text{poly}(\log n, d) \).

We omit this result here due to space constraints.
Acknowledgments
We are grateful to anonymous reviewers for helpful comments, bringing [53] to our attention, and pointing out the recent work [42] that allows us to derandomize the oracle in Section 5. We would like to thank Thatchaphol Saranurak for providing an early manuscript of [62], and Zhijun Zhang for helpful comments on a draft version of this paper.

References


[54] Liam Roditty and Uri Zwick. Dynamic approxi-


