Maintaining Exact Distances under Multiple Edge Failures

Ran Duan
Tsinghua University
Beijing, China
duanran@mail.tsinghua.edu.cn

Hanlin Ren
University of Oxford
Oxford, UK
hanlin.ren@cs.ox.ac.uk

ABSTRACT

We present the first compact distance oracle that tolerates multiple failures and maintains *exact* distances. Given an undirected weighted graph $G = (V, E)$ and an arbitrarily large constant d, we construct an oracle that given vertices $u, v \in V$ and a set of d edge failures D, outputs the *exact* distance between u and v in $G - D$ (that is, G with edges in D removed). Our oracle has space complexity $O(dn^4)$ and query time $d^{O(d)}$. Previously, there were compact *approximate* distance oracles under multiple failures [Chechik, Cohen, Fiat, and Kaplan, SODA'17; Duan, Gu, and Ren, SODA'21], but the best exact distance oracles under d failures require essentially $\Omega(n^d)$ space [Duan and Pettie, SODA'09]. Our distance oracle seems to require $n^{\Omega(d)}$ time to preprocess; we leave it as an open question to improve this preprocessing time.

CCS CONCEPTS

• **Theory of computation → Data structures design and analysis**.

KEYWORDS

Distance sensitivity oracles, dynamic data structures, shortest paths

ACM Reference Format:
Ran Duan and Hanlin Ren. 2022. Maintaining Exact Distances under Multiple Edge Failures. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC '22), June 20–24, 2022, Rome, Italy*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3519935.3520002

1 INTRODUCTION

Real-life networks are dynamic. Sometimes a link or node suffers from a crash failure and has to be removed from the network. Occasionally a new link or node is added to the network. This motivates the study of *dynamic* graph algorithms: algorithms that receive a stream of updates to the graph and needs to simultaneously respond to queries about the *current* graph. The field of dynamic graph algorithms is both classical and vibrant that we will not be able to survey here.

However, in many situations, the network is not "too" dynamic in the sense that the graph will always remain close to a "base" graph. Thus, by preprocessing this base graph, one might obtain better query time bounds than what is possible in the fully dynamic setting. One example is the *d-failure* model: in each query, there is a (small) set of failures (which are either vertices or edges), and we are interested in the graph with the failures removed. After this query was done, the failures are repaired and do not influence the next query.

In this paper, we consider the problem of maintaining distances in the d-failure model. More precisely, given a graph $G = (V, E)$, we want to build an oracle that answers the following queries quickly: given a set of edge failures $D \subseteq E$ with size at most d and two vertices $u, v \in V$, what is the distance from u to v in $G - D$ (i.e., the graph with edges in D removed)?

1.1 Previous Works

The case of $d = 1$ (i.e., only one failure) is well-understood. There is an oracle of size $O(n^2 \log n)$ and query time $O(1)$ that maintains exact distances in a directed graph under one edge or vertex failure [11]. A long line of work [3, 4, 6, 16, 18, 19, 22, 26] has focused on optimising the space or preprocessing time of this oracle.

The case of $d = 2$ was considered by Duan and Pettie [13]: they presented an oracle of size $O(n^2 \log^3 n)$ and query time $O(\log n)$ that maintains exact distances in directed graphs under two vertex or edge failures. Unfortunately, their techniques do not seem to generalise to even 3 failures. They even concluded that "moving beyond dual-failures will require a fundamentally different approach to the problem."

The problem becomes significantly harder when d becomes large. In fact, previous works in this regime had to weaken the query requirements: instead of exact distance queries, they could only handle connectivity queries or *approximate* distance queries.

(1) Pătraşcu and Thorup [21] presented an oracle for handling connectivity queries under d edge failures in an undirected graph. Their oracle has size $\tilde{O}(m)$[1] and query time $\tilde{O}(d)$. Duan and Pettie [14, 15] presented oracles that answer connectivity queries under d *vertex* failures[2], which has size $\tilde{O}(m)$ and query time $\tilde{O}(d^2)$.
(2) Chechik, Langberg, Peleg, and Roditty [8] designed $O(kd)$-approximate distance oracles under d edge failures in an undirected graph, which has size $O(dkn^{1+1/k} \log(nW))$ and query time $\tilde{O}(d)$; here k is an arbitrary constant and W is an upper bound on the edge weights. Bilò, Gualà, Leucci, and Proietti [5] improved the approximation ratio to $(2d + 1)$, with the expense of a larger query time of $\tilde{O}(d^2)$.

[1] $\tilde{O}$ hides polylog(n) factors. In particular, $\tilde{O}(d)$ means $d \cdot$ polylog(n), instead of $d \cdot$ polylog(d).
[2] Edge failures are always no harder than vertex failures, as we can always insert a vertex in the middle of every edge to simulate an edge failure by a vertex failure. However, in many cases [12, 14], dealing with vertex failures requires significantly new ideas compared to edge failures.

STOC '22, June 20–24, 2022, Rome, Italy

ACM ISBN 978-1-4503-9264-8/22/06.
https://doi.org/10.1145/3519935.3520002

(3) If we are willing to tolerate a space complexity bound exponential in d, then we could even achieve $(1+\epsilon)$-approximation for every $\epsilon > 0$. Chechik, Cohen, Fiat, and Kaplan [7] designed $(1+\epsilon)$-approximate distance oracles under d edge failures in undirected graphs, which has size $O(n^2(\log n/\epsilon)^d \cdot d \log W)$ and query time $O(d^5 \log n \log\log W)$. Duan, Gu, and Ren [12] generalised this oracle to also handle vertex failures; their oracle has size $n^{2.01} \cdot (\log n/\epsilon)^{O(d)} \cdot \log W$ and query time $\mathrm{poly}(d, \log n, \log\log W)$.
(4) The harder case of directed weighted graphs was also studied. Weimann and Yuster [26] designed exact distance oracles of size $\tilde{O}(n^{3-\alpha})$ and query time $\tilde{O}(n^{2-2(1-\alpha)/d})$; here $\alpha \in (0,1)$ is an arbitrary parameter. Using an algebraic technique, Brand and Saranurak [25] designed both reachability and exact distance oracles: their reachability oracle has size $\tilde{O}(n^2)$ and query time $O(d^\omega)$, while their distance oracle has size $\tilde{O}(Wn^{2+\alpha})$ and query time $O(Wn^{2-\alpha}d^2 + Wnd^\omega)$. Here $\omega < 2.3728596$ is the matrix multiplication exponent [2, 9, 17, 23, 27].

Exact distances? Despite significant efforts on the d-failure model, our understanding about the setting where *exact* distances need to be maintained is still quite primitive. One reason is that the structure of shortest paths after d failures appears to be very complicated: [13] used heavy case analysis to deal with *two* failures, and three failures appear to be even harder! (We also think this complexity provides further motivation for studying exact distance oracles in the d-failure model, as a good oracle enhances our understanding of the structure of d-failure shortest paths.)

All oracles in the above list could only answer connectivity or approximate distance queries. The only exceptions are the distance oracles in (4), but these oracles have query time polynomial in n and W. Ideally, we want an oracle with query time $\mathrm{poly}(\log n, \log W)$. Thus the following question is open:

Problem 1. Fix a large constant d. Is there a d-failure oracle for handling exact distance queries in undirected graphs with query time $\mathrm{poly}(\log n, \log W)$ and a reasonable size bound?

In fact, before our work, the best d-failure exact distance oracle with a reasonable query time requires size $\tilde{\Omega}(n^d)$ [13], only slightly better than the trivial $O(n^{d+2})$ bound.[3] The trivial bound is obtained as follows: for every set of failures D with size at most d, we store the all-pairs shortest path matrix for the graph $G - D$, which requires $\binom{n}{d} \cdot O(n^2) \le O(n^{d+2})$ space complexity. Duan and Pettie [13] observed that their dual-failure oracle helps shave a factor of about n^2 from this trivial bound: for every set of failures D with size at most $d-2$, we build a dual-failure exact distance oracle for $G - D$ which occupies size $\tilde{O}(n^2)$, and the total space complexity becomes $\binom{n}{d-2} \cdot \tilde{O}(n^2) \le \tilde{O}(n^d)$. However, even the answer to the following problem was unknown:

Problem 2. Is there a 100-failure exact distance oracle for undirected graphs with query time $\mathrm{poly}(\log n, \log W)$ and size $O(n^{99.9})$?

1.2 Our Results

Our main result is an exact distance oracle under d edge failures, for every constant $d \ge 1$.

[3] For simplicity, this particular paragraph only considers vertex failures. The naïve bound for edge failures is $O(m^{d-2}n^2)$ which is even worse than the naïve bound for vertex failures.

Theorem 1.1. *Let $G = (V, E)$ be an undirected weighted graph. For every constant $d \ge 1$, there is an oracle that handles exact distance queries in G under d edge failures. The oracle has size $O(n^4)$ and query time $O(1)$.*

Our oracle is the first one that maintains *exact* distances under multiple (say 100) failures while having reasonable size and query time bounds ($O(n^4)$ and $O(1)$ respectively). In particular, we answer Problems 1 and 2 affirmatively. Unfortunately, we do not know how to preprocess our oracle in less than $n^{\Omega(d)}$ time. We leave it as an open problem to improve the preprocessing time of our oracle.

Our oracle also extends to super-constant d. In this case, our oracle has query time $d^{O(d)}$ and size $O(dn^4)$. We note that an exponential dependence on d also occurs in the best data structures for maintaining $(1+\epsilon)$-approximate distances [7, 12] (albeit in the space complexity bounds instead of the query time bounds).

1.3 Notation

Let $G = (V, E, w)$ be an undirected graph with edge weights $w : E \to \mathbb{R}$. When $D \subseteq E$, we use $G - D$ to denote the graph G with edges in D removed. For a graph H and two vertices $u, v \in V(H)$, $\pi_H(u, v)$ denotes the shortest path from u to v in H, and $|\pi_H(u, v)|$ denotes the length of this shortest path. When $H = G$ is the input graph, we may omit the subscript H, i.e., $\pi(u, v) = \pi_G(u, v)$. Although this paper only considers undirected graphs, the paths will be directed, i.e., $\pi_H(u, v)$ is a path *from u to v* and u (or v) is the first (or last) vertex on it.

We assume that the shortest paths in G are unique. If not, we could randomly perturb the edge weights of G by a small value; the correctness follows from the isolation lemma [20, 24]. Alternatively, we could use a method described in [10, Section 3.4] to obtain unique shortest paths. We omit the details here.

We will also consider shortest path trees (in the input graph G). For vertices $v, v' \in V$, T_v denotes the shortest path tree rooted at v, and $T_v(v')$ denotes the subtree of T_v rooted at v' (which contains all vertices w such that v' is on the path $\pi(v, w)$).

2 AN OVERVIEW OF OUR ORACLE

In this section, we present a high-level overview of our exact distance oracle.

A recursive approach. Our starting point is the following structural theorem for the shortest paths in an undirected graph with d edge failures [1, Theorem 2]. (See also Theorem 3.1.)

Theorem 2.1. *Let G be an undirected graph, D be a set of d edge failures. Any shortest path in $G - D$ can be decomposed into the concatenation of at most $d+1$ shortest paths in G interleaved with at most d edges.*

Given a query (u, v, D), we want to find $P_{\mathsf{ans}} = \pi_{G-D}(u, v)$. From Theorem 2.1, P_{ans} is divided into $d+1$ segments where each segment is a shortest path in G. Our strategy is to find an arbitrary vertex $w \in P_{\mathsf{ans}}$ *which is neither on the first nor on the last segment*, recursively find $\pi_{G-D}(u, w)$ and $\pi_{G-D}(w, v)$, and concatenate these two paths. If P_{ans} can be decomposed into k segments and w is neither on the first nor on the last segment, then both $\pi_{G-D}(u, w)$ and $\pi_{G-D}(w, v)$ can be decomposed into at most $d-1$ segments. It follows that the recursion depth is at most $d+1$.

We do not know how to find a single vertex w, but we are able to find a "hitting set" consisting of $\text{poly}(d)$ many vertices w such that at least one w sits on P_{ans} and is neither on the first nor on the last segment of P_{ans}. Therefore, the query time is $\text{poly}(d)^d = d^{O(d)}$.

Finding a hitting set. Now, it remains to design a procedure for finding such a hitting set. It suffices to find a set $H \subseteq V$ such that:

(I) there is a vertex $w \in H$ that lies on P_{ans};
(II) $|H|$ should be small; and
(III) every vertex $w \in H$ that lies on P_{ans} does not lie on the first or the last segment of P_{ans}.

As a warm-up, we present a (very simple!) hitting set satisfying (I) and (II). Of course it is (III) that enables us to upper bound the recursion depth; we will address (III) later.

For every $u, v \in V$, we simply let $\mathcal{D}[u, v]$ be the set of d edge failures *whose removal maximises the distance from u to v*. Note that $\mathcal{D}[u, v]$ does not depend on D and therefore can be preprocessed in advance. Since $|\mathcal{D}[u, v]| \le d$, (II) is true. Now we claim that either $\mathcal{D}[u, v]$ hits P_{ans}, or $\pi_{G-\mathcal{D}[u,v]}(u, v)$ and P_{ans} are exactly the same path. Indeed, if $\mathcal{D}[u, v]$ does not hit P_{ans}, then P_{ans} is no shorter than the path $\pi_{G-\mathcal{D}[u,v]}(u, v)$, which means D should be *as good as* the best candidate for $\mathcal{D}[u, v]$! And $\pi_{G-\mathcal{D}[u,v]}(u, v)$ and P_{ans} should coincide in this case.[4]

We regard the latter case (i.e. $\pi_{G-\mathcal{D}[u,v]}(u, v) = P_{\text{ans}}$) as "trivial" since it can be handled in preprocessing. Therefore, in this informal overview, we will ignore this case and simply say that $\mathcal{D}[u, v]$ is a hitting set of P_{ans}.

Unfortunately, we are not aware of any fast algorithm for computing (any reasonable approximation of) $\mathcal{D}[u, v]$, therefore we are currently unable to obtain a fast preprocessing algorithm for our data structure.

Achieving (III) via cleanness. Suppose we have found a hitting set H satisfying (I) and (II) but not (III). Without loss of generality, suppose that there is a vertex $w \in H$ on the first segment of P_{ans}. This implies that $\pi(u, w)$ is intact from failures (as it lies entirely inside the first segment of P_{ans}). Our first insight is that if w is *"u-clean"*, then we could find another hitting set satisfying (I) and (II) and *avoiding the first segment of P_{ans}*.

The definition of cleanness is as follows: We say w is *u-clean* if both $\pi(u, w)$ and $T_u(w)$ are intact from failures.[5] (Recall that T_u is the shortest path tree rooted at u, and $T_u(w)$ is the subtree of T_u with root w.) Now suppose w is u-clean and P_{ans} goes through w. Let $\mathcal{D}_\star$ be the maximiser of $|\pi_{G-D'}(u, v)|$ such that

$$\textit{both } \pi(u, w) \textit{ and } T_u(w) \textit{ are intact from } D'. \qquad (\sigma)$$

Again, $\mathcal{D}_\star$ depends on u, v, w but not on D, so it can be preprocessed in advance. As w is u-clean, D also satisfies (σ). Therefore $\mathcal{D}_\star$ hits P_{ans} by the above reasoning.

Now consider a vertex w' incident to some edge in $\mathcal{D}_\star$ and suppose that P_{ans} also goes through w'. If $\pi(u, w')$ is intact from D, then either $\pi(u, w')$ does not go through w (in this case P_{ans} does not go through w either) or w' is in $T_u(w)$ (violating (σ)), a contradiction. Therefore we only need to consider vertices $w' \in \mathcal{D}_\star$ such that $\pi(u, w')$ is not intact from D; such vertices w' can never hit the first segment of P_{ans}.

One can similarly define the notion of v-cleanness (which we omit here). The above discussion generalises to the following statement: If we know two vertices u' and v' such that u' is u-clean, v' is v-clean, and P_{ans} goes through both u' and v', then we can find a hitting set satisfying all three conditions above. Note that we need to store a set $\mathcal{D}_\star$ for each possible (u, v, u', v'), therefore our oracle requires $O(dn^4)$ space complexity.

Finding a clean vertex. Now, the problem reduces to finding u-clean and v-clean vertices that hit P_{ans}. For simplicity, in this overview, we consider the scenario where we already know a v-clean vertex v' that is on P_{ans}, and want to find a u-clean vertex u' that also hits P_{ans}. We believe this scenario already captures our core technical ideas.

Consider the following naïve attempt. Suppose we have a vertex u' but $T_u(u')$ contains some failures. Our goal is to "push" u' to a deeper vertex in T_u so that eventually $T_u(u')$ will be intact from D. Let $\mathcal{D}_\star$ be the maximiser of $|\pi_{G-D'}(u, v)|$ such that

$$\textit{all of } \pi(v', v), T_v(v'), \textit{ and } \pi(u, u') \textit{ are intact from } D'. \qquad (\tau_{\text{naïve}})$$

Consider any vertex w incident to some edge in $\mathcal{D}_\star$. It turns out that if w is not a *strict* descendant of u' (i.e. $w \notin T_u(u')$ or $w = u'$) then we can deal with w easily. If w is a strict descendant of u', then we assign $u' \leftarrow w$ (i.e. push u' down to w) and repeat. We have made some progress as we pushed u' to w and $T_u(w)$ is a subset $T_u(u')$. But after how many steps would $T_u(w)$ become intact from D? It seems that we might need $\Omega(n)$ steps before we push u' down to some vertex w which is u-clean.

Our second insight is the following modification to $(\tau_{\text{naïve}})$. Let u_{LCA} be the least common ancestor of all failures in $T_u(u')$, i.e., the lowest vertex in T_u which is the ancestor of (both endpoints of) every failure in $T_u(u')$. If D is intact from $\pi(u, u')$, then D *should also be intact from* $\pi(u, u_{\text{LCA}})$. Now, let $\mathcal{D}_\star$ be the maximiser of $|\pi_{G-D'}(u, v)|$ such that

$$\textit{all of } \pi(v', v), T_v(v'), \textit{ and } \pi(u, \underline{u_{\text{LCA}}}) \textit{ are intact from } D'. \qquad (\tau)$$

Note that D still satisfies (τ) which means $\mathcal{D}_\star$ is still a valid hitting set. We can push u' down to some vertex w which is a strict descendant of u_{LCA}. Now comes the crucial observation: the number of failures in $T_u(w)$ is *strictly smaller* than the number of failures in $T_u(u')$. Since there are at most d failures in $T_u(u')$, it takes at most d steps of "pushing u' down" before u' becomes u-clean (i.e. $T_u(u')$ becomes intact from D).

We remark that in the formal proof in Section 4.2.2 there is no implementation of "pushing u' down". Instead, we enumerate the portion of T_u where the last step of pushing happens (i.e., u' "becomes" u-clean). Nevertheless, the two formulations are equivalent, and we find the above description more intuitive.

3 AN EXACT DISTANCE ORACLE FOR EDGE FAILURES

In this section, we describe the framework for our exact distance oracle that handles d edge failures and prove Theorem 1.1.

As mentioned in Section 2, we use the structural theorem of shortest paths under edge failures in [1]. We say that a path is

[4] Note that $\mathcal{D}[u, v]$ is a set of *edges*. To find a hitting set consisting of vertices, simply take both endpoints of every edge in $\mathcal{D}[u, v]$. The hitting set still have size $O(d)$.
[5] To be more precise, "$T_u(w)$ is intact from failures" means that *every vertex in $T_u(w)$ is not incident to any failed edge.*

a *k-decomposable path* if it is the concatenation of at most $k+1$ shortest paths in G, interleaved with at most k edges. (Here two adjacent original shortest paths in G can be directed concatenated or linked by an edge.) We have:

Theorem 3.1 (Theorem 2 of [1]). *For any set D of d edge failures in the graph and any vertices $u, v \in V$, the path $\pi_{G-D}(u,v)$ is a d-decomposable path.*

For a set D of d edge failures and vertices $u, v \in V$, we define $\mathrm{rank}_{G-D}(u,v)$ as the smallest number i such that $\pi_{G-D}(u,v)$ is an i-decomposable path. For example, $\mathrm{rank}_{G-D}(u,v) = 0$ if and only if the shortest u-v path contains no failures. Thus, the conclusion of Theorem 3.1 can be interpreted as "$\mathrm{rank}_{G-D}(u,v) \le |D|$."

Our query algorithm relies on a subroutine HitSet(u,v,D), whose details will be given in Section 4. Given $u, v \in V$ and a set of d edge failures D, the output of HitSet(u,v,D) consists of an upper bound L of $|\pi_{G-D}(u,v)|$ and a set of vertices $H \subseteq V$, such that the following holds.

(a) Either $|\pi_{G-D}(u,v)| = L$, or $\pi_{G-D}(u,v)$ goes through some vertex in H.
(b) $|H| \le O(d^6)$.
(c) For every vertex $w \in H$, both $\pi(u,w)$ and $\pi(w,v)$ contain failures in D.

Note that Item (c) ensures the following property:

Claim 3.2. *Let $u, v \in V$, D be a set of d edge failures, and $r = \mathrm{rank}_{G-D}(u,v)$. Suppose w is a vertex in H that is also on $\pi_{G-D}(u,v)$. Then $\mathrm{rank}_{G-D}(u,w) \le r-1$ and $\mathrm{rank}_{G-D}(w,v) \le r-1$.*

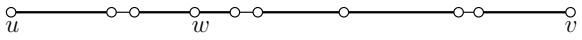

Figure 1: An example of Claim 3.2. Here $\mathrm{rank}_{G-D}(u,v) = 4$ and $\pi_{G-D}(u,v)$ is decomposed into 5 shortest paths in G (depicted as bold segments) interleaved with 3 ($\le \mathrm{rank}_{G-D}(u,v)$) edges.

Proof. We decompose $\pi_{G-D}(u,v)$ into $r+1$ shortest paths interleaved with at most r edges. It is easy to see that w is neither on the first shortest path nor on the last shortest path, as otherwise either $\pi(u,w)$ or $\pi(w,v)$ is intact from D, contradicting our assumption that $w \in H$.

Since w is not on the last shortest path, the portion of the decomposition from u to w includes at most r shortest paths interleaved with at most $r-1$ edges, therefore $\mathrm{rank}_{G-D}(u,w) \le r-1$. Similarly, since w is not on the first shortest path, $\mathrm{rank}_{G-D}(w,v) \le r-1$. □

We illustrate the query algorithm in Algorithm 1. Roughly speaking, the idea is to enumerate a hitting vertex $w \in H$ and recursively find $\pi_{G-D}(u,w)$ and $\pi_{G-D}(w,v)$.

In Section 4, we will show that an invocation of HitSet(u,v,D) takes $\mathrm{poly}(d)$ time. Therefore, our query algorithm runs in $O(d^6)^d \cdot \mathrm{poly}(d) \le d^{O(d)}$ time. The following theorem demonstrates the correctness of our query algorithm.

Theorem 3.3. *Assuming the correctness of the HitSet structure, the query algorithm is correct.*

Algorithm 1 Query Algorithm for the Exact Distance Oracle

```
1: function Query-r(u, v, D, r)                ▷ We assume that rank_{G-D}(u,v) ≤ r.
2:     if π(u,v) ∩ D = ∅ then return |π(u,v)|
3:     if r = 0 then return +∞
4:     (L, H) ← HitSet(u, v, D)
5:     ans ← L
6:     for each w in H do
7:         ans ← min{ans, Query-r(u, w, D, r − 1) + Query-r(w, v, D, r − 1)}
8:     return ans
9: function Query(u, v, D)
10:    return Query-r(u, v, D, |D|)
```

Proof. We show that for every u, v, D, r, if $\mathrm{rank}_{G-D}(u,v) \le r$, then Query-r$(u,v,D,r) = |\pi_{G-D}(u,v)|$. The theorem follows from Theorem 3.1.

Actually, it is easy to see that Query-r$(u,v,D,r) \ge |\pi_{G-D}(u,v)|$ as we could always construct a u-v path in $G-D$ with length Query-r(u,v,D,r). Therefore it suffices to show that

$$\text{Query-r}(u,v,D,r) \le |\pi_{G-D}(u,v)|.$$

We use induction on r. Our assertion is clearly true for $r = 0$. Let $r \ge 1$ and $(L,H) =$ HitSet(u,v,D). If $|\pi_{G-D}(u,v)| = L$ then we are done, as Query-r$(u,v,D,r) \le L$ in this case. Otherwise by Item (a) in the correctness of HitSet, $\pi_{G-D}(u,v)$ hits some vertex $w \in H$. By Claim 3.2, $\mathrm{rank}_{G-D}(u,w) \le r-1$ and $\mathrm{rank}_{G-D}(w,v) \le r-1$. By induction, we have that

$$\text{Query-r}(u,w,D,r-1) = |\pi_{G-D}(u,w)|$$
$$\text{and} \quad \text{Query-r}(w,v,D,r-1) = |\pi_{G-D}(w,v)|.$$

It follows that

$$\text{Query-r}(u,v,D,r) \le |\pi_{G-D}(u,w)| + |\pi_{G-D}(w,v)| = |\pi_{G-D}(u,v)|.$$

□

4 THE HitSet STRUCTURE

In this section, we describe the implementation of HitSet. Recall that given $u, v \in V$ and a set of d edge failures D, it should output an upper bound L of $|\pi_{G-D}(u,v)|$ and a set of vertices $H \subseteq V$, such that the following items hold.

(a) Either $|\pi_{G-D}(u,v)| = L$, or $\pi_{G-D}(u,v)$ goes through some vertex in H.
(b) $|H| \le O(d^6)$.
(c) For every vertex $w \in H$, both $\pi(u,w)$ and $\pi(w,v)$ contain failures in D.

Warm-up. Suppose that we drop Item (c) above, then it is easy to present a data structure for HitSet with space complexity $O(dn^2)$. For every two vertices $u, v \in V$, let $\mathcal{D}[u,v]$ be the set of d edge failures that maximises $|\pi_{G-\mathcal{D}[u,v]}(u,v)|$. The data structure simply stores $\mathcal{D}[u,v]$ and $\ell_{\max}(u,v) = |\pi_{G-\mathcal{D}[u,v]}(u,v)|$. In each query HitSet$(u,v,D)$, for every edge $e \in \mathcal{D}[u,v]$, we arbitrarily pick an endpoint of e and add it into H. Then we return $L = \ell_{\max}(u,v)$ and H. Item (b) holds since $|\mathcal{D}[u,v]| \le d$. Therefore it suffices to prove that Item (a) holds:

Claim 4.1. *For every set of d edge failures D, either $|\pi_{G-D}(u,v)| = \ell_{\max}(u,v)$, or $\pi_{G-D}(u,v)$ goes through some vertex in H.*

Proof. Actually, we show that either $|\pi_{G-D}(u,v)| = \ell_{\max}(u,v)$ or $\pi_{G-D}(u,v)$ goes through some edge in $\mathcal{D}[u,v]$. Suppose that $\pi_{G-D}(u,v)$ does not intersect $\mathcal{D}[u,v]$. That is, $\pi_{G-D}(u,v)$ is a valid path from u to v that does not go through $\mathcal{D}[u,v]$. It follows that

$$|\pi_{G-D}(u,v)| \geq |\pi_{G-\mathcal{D}[u,v]}(u,v)| = \ell_{\max}(u,v).$$

However, we also have $|\pi_{G-\mathcal{D}[u,v]}(u,v)| \geq |\pi_{G-D}(u,v)|$ by the definition of $\mathcal{D}[u,v]$. Therefore

$$|\pi_{G-D}(u,v)| = \ell_{\max}(u,v). \qquad \square$$

The warm-up case demonstrates that it is easy to satisfy Items (a) and (b). All technical complications introduced in the rest of this section deals with Item (c).

4.1 The Data Structure

For every four vertices $u, v, u', v' \in V$ and two Boolean variables $b_1, b_2 \in \{0,1\}$, our data structure contains a size-d edge set

$$\mathcal{D}[u, v, u', v', b_1, b_2],$$

corresponding to the scenario where we want to find $\pi_{G-D}(u,v)$ (where D is a set of failures given in the query), and we know two intermediate vertices u', v' satisfying the following properties:

(i) The paths $\pi(u,u')$ and $\pi(v',v)$ are intact from D.
(ii) We *assume* that $\pi_{G-D}(u,v)$ goes through both u' and v'; in other words, $\pi(u,u')$ is a prefix of $\pi_{G-D}(u,v)$, and $\pi(v',v)$ is a suffix of $\pi_{G-D}(u,v)$.

(An intuitive interpretation of u' and v' is as follows. We are trying to find a hitting vertex w such that both $\pi(u,w)$ and $\pi(w,v)$ intersect D, so we can add w into our hitting set H; u' and v' represent our failed attempts, i.e., hitting vertices w where $\pi(u,w)$ or $\pi(w,v)$ did not happen to intersect D.)

The meaning of Boolean variables b_1 and b_2 are as follows. If $b_1 = 1$, then we require that $T_u(u') \cap V(D) = \emptyset$, where $V(D)$ is the set of vertices incident to some failure in D. (Recall that T_u is the shortest path tree rooted at u, and $T_u(u')$ is the subtree of T_u rooted at u'.) If $b_1 = 0$, we do not impose any condition on $T_u(u') \cap V(D)$. Similarly, if $b_2 = 1$ then $T_v(v') \cap V(D) = \emptyset$, while if $b_2 = 0$ then we do not impose any condition on $T_v(v') \cap V(D)$.

Naturally, we define $\mathcal{D}[u, v, u', v', b_1, b_2]$ as the set D' that maximises $|\pi_{G-D'}(u,v)|$, subject to Item (i) and the conditions imposed by b_1 and b_2. For example, $\mathcal{D}[u, v, u', v', 0, 1]$ is the maximiser of $|\pi_{G-D'}(u,v)|$ among all size-d edge sets D' where (See also Fig. 2)

$$\pi(u,u') \cap D' = \emptyset, \pi(v',v) \cap D' = \emptyset, \text{ and } T_v(v') \cap V(D') = \emptyset.$$

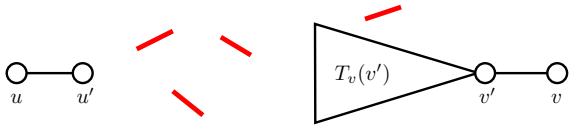

Figure 2: An illustration of $\mathcal{D}[u, v, u', v', 0, 1]$. The red bold edges are a possible set of edge failures D' that does not intersect $\pi(u,u')$, $\pi(v',v)$, and $T_v(v')$.

Our data structure occupies $O(dn^4)$ space. It is easy to see that our data structure can be preprocessed in time $m^{d+O(1)}$. We leave it as an open problem to improve this preprocessing time.

4.2 The HitSet Algorithm

In what follows, we say that a vertex $w \in V$ is *u-clean* if $\pi(u,w) \cap D = \emptyset$ and $T_u(w) \cap V(D) = \emptyset$. Similarly, we say that a vertex $w \in V$ is *v-clean* if $\pi(w,v) \cap D = \emptyset$ and $T_v(w) \cap V(D) = \emptyset$.

We present HitSet from special cases to the most general case.

- In Case I, we assume that we know two "helper" vertices u', v' such that u' is u-clean, v' is v-clean, and $\pi_{G-D}(u,v)$ goes through both u' and v'. This case is similar to the warm-up case and is essentially one invocation of the data structure $\mathcal{D}[\cdot]$.
- In Case II, we assume that we know *one* "helper" vertex v' (such that v' is v-clean and lies on $\pi_{G-D}(u,v)$), but we need to find the other "helper" vertex (u'). We will find a small number of candidate vertices u', thus reducing this case to Case I. *Most of our new techniques will appear in this case.*
- Case III is the most general case where we do not have any "helper" vertices and need to find them on our own. Nevertheless, using techniques similar to Case II, we can still find a small number of such "helper" vertices and reduce this case to Case II.

4.2.1 Case I. In this case, we assume that we already know vertices $u', v' \in V$ such that u' is u-clean, v' is v-clean, and $\pi_{G-D}(u,v)$ goes through both u' and v'. (See Fig. 3.)

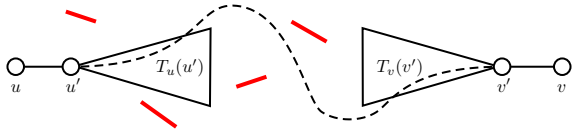

Figure 3: An illustration of Case I. Here, the red bold edges denote D. As $\pi(u,u') \cap D = \emptyset$ and $T_u(u') \cap V(D) = \emptyset$, u' is u-clean. Similarly, v' is v-clean. The dashed path denotes $\pi_{G-D}(u,v)$, which goes through both u' and v'.

This case can be solved similarly as in the warm-up case. Let $\mathcal{D}_\star = \mathcal{D}[u, v, u', v', 1, 1]$, then $\mathcal{D}_\star$ is the maximiser of $|\pi_{G-D'}(u,v)|$ over all size-d edge sets D' such that

$$\pi(u,u'), \pi(v',v), T_u(u'), \text{and } T_v(v') \text{ are intact from } D'. \qquad (\alpha)$$

Let $L = |\pi_{G-\mathcal{D}_\star}(u,v)|$ and

$H = \{w \in V(\mathcal{D}_\star) : \text{both } \pi(u,w) \text{ and } \pi(w,v) \text{ contain failures in } D\}$.

It is easy to see that Items (b) and (c) hold, so it suffices to show Item (a), i.e.:

Claim 4.2. *Either $|\pi_{G-D}(u,v)| = L$ or $\pi_{G-D}(u,v)$ goes through some vertex in H.*

Proof. We first show that if $\pi_{G-D}(u,v)$ goes through some edge in $\mathcal{D}_\star$, then it also goes through some vertex in H. Suppose that $\pi_{G-D}(u,v)$ goes through some edge $e = (x,y) \in \mathcal{D}_\star$, we claim that $\pi(u,x)$ is not intact from D. Since $\pi_{G-D}(u,v)$ goes through

u' and $\pi(u,u')$ is intact from D, $\pi(u,u')$ coincides with the path from u to u' in T_u. Suppose $\pi(u,x)$ is also intact from D, then x has to be either an ancestor or a descendant of u' in T_u. Since $T_u(u') \cap V(\mathcal{D}_\star) = \emptyset$, x cannot be a descendant of u'. Therefore, x is a strict ancestor of u'. However, as e is an incident edge of x in the path $\pi_{G-D}(u,v)$, it has to be on the path $\pi(u,u')$, which contradicts the fact that $\pi(u,u') \cap \mathcal{D}_\star = \emptyset$. (See Fig. 4.)

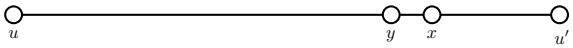

Figure 4: Illustration of Claim 4.2.

Therefore, $\pi(u,x)$ cannot be intact from D. Similarly, $\pi(x,v)$ cannot be intact from D either. It follows that $x \in H$.

Now the argument is essentially the same as Claim 4.1. Suppose that $\pi_{G-D}(u,v)$ does not go through any edge in $\mathcal{D}_\star$, then

$$|\pi_{G-D}(u,v)| \geq |\pi_{G-\mathcal{D}_\star}(u,v)| = L.$$

However, $\mathcal{D}_\star$ is the maximiser of $|\pi_{G-D'}(u,v)|$ over all size-d edge sets D' satisfying the condition (α). Since u' is u-clean and v' is v-clean, D also satisfies (α), thus

$$|\pi_{G-D}(u,v)| = |\pi_{G-\mathcal{D}_\star}(u,v)| = L. \qquad \square$$

4.2.2 Case II. In this case, we assume that we know a vertex $v' \in V$ such that v' is v-clean, and $\pi_{G-D}(u,v)$ goes through v'. The goal of this case is to find a small number of candidates u', such that every u' is u-clean and $\pi_{G-D}(u,v)$ goes through one of these u'. In this way, we can reduce this case to Case I.

First, we add a new vertex u_{root}, add an edge (u_{root}, u) to connect it to T_u, and make u_{root} the root of T_u. This step is solely for convenience.

Denote T_{induced} the induced subtree of $V(D) \cup \{u_{\text{root}}\}$ over T_u, i.e. an edge is in T_{induced} if it is on some path between two vertices in $V(D) \cup \{u_{\text{root}}\}$. We say a vertex v is a *key* vertex if either $v \in V(D) \cup \{u_{\text{root}}\}$ or the degree of v in T_{induced} is at least 3. Let Key be the set of key vertices, then $|\text{Key}| \leq O(d)$. By contracting every non-key vertex in T_{induced} (note that these vertices have degree exactly 2), we obtain a smaller tree T_{key} over Key where each edge (x,y) in T_{key} corresponds to a path from x to y in T_{induced}; here x and y are key vertices and all the intermediate vertices on the path have degree 2.

Let e_Δ be the last edge on $\pi_{G-D}(u,v)$ such that the portion from u to e_Δ in $\pi_{G-D}(u,v)$ is entirely in T_{induced}. Note that e_Δ always exists since we added the auxiliary u_{root}. (In particular, if $\pi_{G-D}(u,v)$ does not intersect T_{induced} at all, we assume e_Δ is the edge between u_{root} and u.)

We enumerate an edge $(p,c) \in E(T_{\text{key}})$ with the hope that e_Δ is on the path from p to c in T_{induced}. Note that there are $O(d)$ possible choices of (p,c). Let $\mathcal{D}_\star = \mathcal{D}[u,v,c,v',0,1]$, then $\mathcal{D}_\star$ maximises $|\pi_{G-\mathcal{D}_\star}(u,v)|$ over all size-d sets of edge failures such that

$$\mathcal{D}_\star \cap \pi(u,c) = \emptyset, \mathcal{D}_\star \cap \pi(v',v) = \emptyset, \text{ and } V(\mathcal{D}_\star) \cap T_v(v') = \emptyset. \qquad (\beta)$$

If $D \cap \pi(u,c) \neq \emptyset$, then we discard the edge (p,c). This is because the following claim shows that e_Δ cannot appear on the path from p to c:

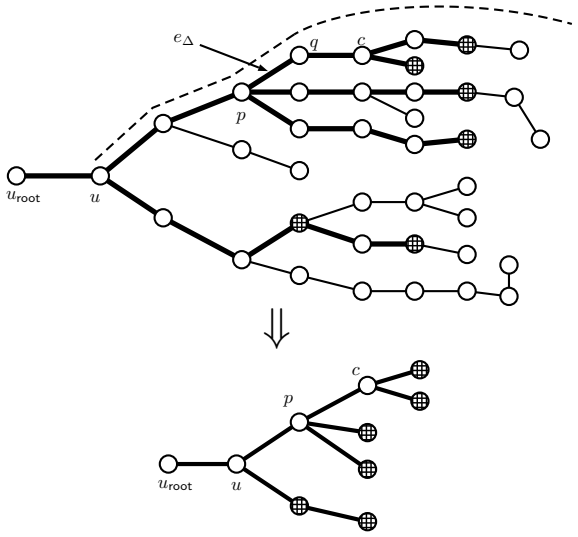

Figure 5: Top: a possible shortest path tree T_u. Hatched vertices are vertices in $V(D)$, and bold edges are edges in T_{induced}. The dash curve corresponds to $\pi_{G-D}(u,v)$, and e_Δ is the edge between p and q. Bottom: the corresponding T_{key}. Note that (p,c) is the edge in T_{key} such that e_Δ is on the path from p to c in T_{induced}.

Claim 4.3. *If $D \cap \pi(u,c) \neq \emptyset$, then e_Δ cannot appear on the path from p to c.*

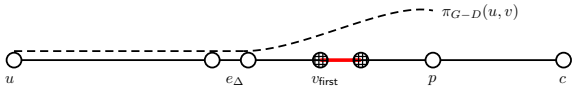

Figure 6: Illustration of Claim 4.3.

Proof. Let v_{first} be the first vertex on $\pi(u,c)$ which is incident to a failed edge in D on $\pi(u,c)$. Since $\pi_{G-D}(u,v)$ avoids the failed edge on $\pi(u,c)$, e_Δ has to be before v_{first}. On the other hand, since $v_{\text{first}} \in V(D) \subseteq \text{Key}$, v_{first} does not lie after p (as otherwise there will be some key vertices between p and c). This means that e_Δ is strictly before p on the path $\pi(u,c)$. $\square$

If $D \cap \pi(u,c) = \emptyset$ then D also satisfies (β). It is now valid to update

$$L \leftarrow \min\{L, |\pi_{G-\mathcal{D}_\star}(u,v)|\}.$$

By the same reasoning as Claim 4.1, if $|\pi_{G-D}(u,v)| < |\pi_{G-\mathcal{D}_\star}(u,v)|$, then $\pi_{G-D}(u,v)$ should go through some edge in $\mathcal{D}_\star$.

Now we construct a set Helper of candidate "helper" vertices u' by examining every edge $e \in \mathcal{D}_\star \setminus D$ one by one. Suppose e is an edge between x and y:

(Case i) If $\pi(x,v) \cap D = \emptyset$ or $\pi(y,v) \cap D = \emptyset$, we discard e. The reason is that $\pi_{G-D}(u,v)$ cannot go through e. Otherwise, suppose w.l.o.g. that $\pi(x,v) \cap D = \emptyset$. Since $\pi_{G-D}(u,v)$ goes through e, in particular it also goes through x. It follows that $\pi(x,v)$ is a suffix of $\pi_{G-D}(u,v)$.

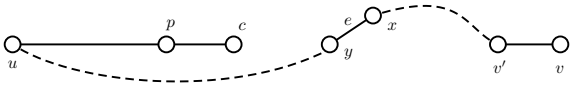

Figure 7: Case II.i.

Figure 8: Case II.iv.

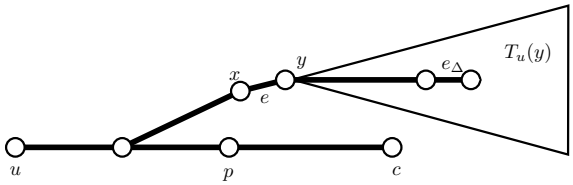

Figure 9: Case II.vi.

Note that $\pi(v', v)$ is also a suffix of $\pi_{G-D}(u, v)$. Therefore x is either an ancestor or a descendant of v' in T_v.

- If x is a descendant of v' in T_v, then x is both in $V(\mathcal{D}_\star)$ and $T_v(v')$, violating (β).
- If x is an ancestor of v' in T_v, then e has to be on the path $\pi(v', v)$ in order for $\pi_{G-D}(u, v)$ to go through e, but this also violates (β).

(Case ii) Otherwise, if $\pi(u, x) \cap D \neq \emptyset$, we add x into H.
Note that in this case, both $\pi(u, x)$ and $\pi(x, v)$ intersect D, therefore it is safe to add x into H.

(Case iii) Otherwise, if $\pi(u, y) \cap D \neq \emptyset$, we add y into H.
This case is similar as (Case ii).

(Case iv) Otherwise, if e is not a tree edge in T_u, we discard e.
This is because in this case, both $\pi(u, x)$ and $\pi(u, y)$ are intact from D, and $\pi_{G-D}(u, v)$ does not need to go through e at all. (For example, if $\pi_{G-D}(u, v)$ goes through e and x appears just before y, then $\pi_{G-D}(u, v)$ should use the path $\pi(u, y)$ instead of the concatenation of $\pi(u, x)$ and e.)

(Case v) Otherwise, e is a tree edge in T_u. W.l.o.g. we assume that x is the parent of y in T_u. If $T_u(y) \cap V(D) = \emptyset$, we add y into Helper.
Note that y is u-clean in this case, so it is safe to add it into Helper.

(Case vi) Otherwise, we discard e.
We need to prove that it is valid to discard e in this case, i.e. $\pi_{G-D}(u, v)$ cannot go through e. Note that since $T_u(y) \cap V(D) \neq \emptyset$, y lies on T_{induced} (i.e. has non-zero degree in T_{induced}). Since $\pi(u, y) \cap D = \emptyset$, if $\pi_{G-D}(u, v)$ goes through y, then $\pi(u, y)$ must be a prefix of $\pi_{G-D}(u, v)$. It follows that e_Δ is either equal to e or in the subtree $T_u(y)$. Since (β) implies that e is not on the path $\pi(u, c)$, e_Δ cannot be on the path $\pi(p, c)$ either, a contradiction.

Summary. In Case II, we first construct the trees T_{induced} and T_{key} in $O(d)$ time. Then we enumerate an edge $(p, c) \in E(T_{\text{key}})$. Let $D_\star = \mathcal{D}[u, v, c, v', 0, 1]$, then for each edge $e \in D_\star \setminus D$, according to the above case analysis, we either discard e, add a vertex into H, or add a vertex into Helper. Note that for every $u' \in$ Helper, (D, u', v') satisfies (α).

After enumerating all edges in $E(T_{\text{key}})$, we have that $|H| \leq O(d^2)$ and $|\text{Helper}| \leq O(d^2)$. Then for each $u' \in$ Helper, we invoke the algorithm for Case I where we assume that $\pi_{G-D}(u, v)$ goes through both u' and v'. Each invocation returns a hitting set of size $O(d)$ and an upper bound L of $|\pi_{G-D}(u, v)|$. Finally, we let L be the smallest upper bound found during the entire execution of the algorithm, and H be the union of all hitting sets (which has size $O(d^3)$). It is easy to see that Case II takes $O(d^3)$ time.

4.2.3 Case III. Case III is the most general case: We only know the query (u, v, D) but no "helper" vertices u' or v'. The goal is to find a few intermediate vertices w which are either u-clean or v-clean, such that $\pi_{G-D}(u, v)$ goes through one of the vertices w.

We construct the trees T^u_{induced}, T^u_{key}, T^v_{induced}, and T^v_{key}. Let e_Δ be the last edge on $\pi_{G-D}(u, v)$ such that the portion from u to e_Δ on $\pi_{G-D}(u, v)$ is entirely in T^u_{induced}, and e_∇ be the first edge such that the portion from e_∇ to v on $\pi_{G-D}(u, v)$ is entirely in T^v_{induced}.

We enumerate edges $(p^u, c^u) \in E(T^u_{\text{key}})$ and $(c^v, p^v) \in E(T^v_{\text{key}})$ such that e_Δ is on the path from p^u to c^u in T^u_{induced}, and e_∇ is on the path from c^v to p^v in T^v_{induced}. Note that there are $O(d^2)$ possible choices of (p^u, c^u) and (p^v, c^v).

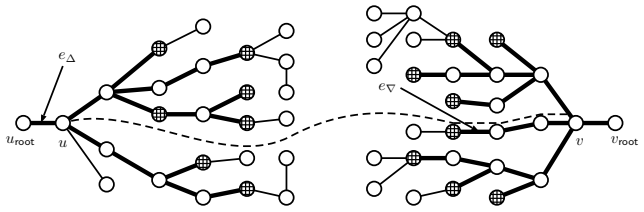

Figure 10: The trees T^u_{induced} (left) and T^v_{induced} (right). Again, the hatched vertices are vertices in $V(D)$, and the dashed curve corresponds to $\pi_{G-D}(u, v)$. Note that in this particular figure, e_Δ coincides with the edge between u_{root} and u.

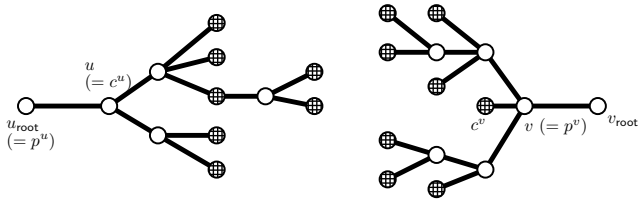

Figure 11: The trees T^u_{key} (left) and T^v_{key} (right).

Now let

$$\mathcal{D}_\star = \mathcal{D}[u, v, c^u, c^v, 0, 0],$$

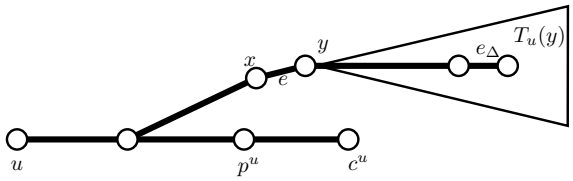

Figure 12: Case III.iii.b.

then $\mathcal{D}_\star$ maximises $|\pi_{G-\mathcal{D}_\star}(u,v)|$ over all size-d set of edge failures such that

$$\mathcal{D}_\star \cap \pi(u,c^u) = \emptyset, \text{ and } \mathcal{D}_\star \cap \pi(c^v,v) = \emptyset. \qquad (\gamma)$$

By Claim 4.3, if $D \cap \pi(u,c^u) \neq \emptyset$, then e_Δ cannot appear in the path from p^u to c^u. Similarly, if $D \cap \pi(c^v,v) \neq \emptyset$, then e_∇ cannot appear in the path from c^v to p^v. Therefore we may assume D satisfies Eq. (γ), as otherwise we can discard (p^u,c^v) and (p^v,c^v). This means that it is safe to update

$$L \leftarrow \min\{L, |\pi_{G-\mathcal{D}_\star}(u,v)|\}.$$

If $|\pi_{G-D}(u,v)| < |\pi_{G-\mathcal{D}_\star}(u,v)|$, then $\pi_{G-D}(u,v)$ should go through some edge in $\mathcal{D}_\star \setminus D$. Now we compute a hitting set H and a set Helper of "helper" vertices by inspecting every edge in $\mathcal{D}_\star \setminus D$. In particular, we enumerate this edge $e=(x,y) \in \mathcal{D}_\star \setminus D$, and assume that x appears before y on the path $\pi_{G-D}(u,v)$. (That is, every edge (x,y) is considered twice, once for (x,y) and once for (y,x).)

(Case i) Suppose that $\pi(u,x) \cap D = \emptyset$ and $\pi(y,v) \cap D = \emptyset$. We can immediately update $L \leftarrow \min\{L, |\pi(u,x)|+w(e)+|\pi(y,v)|\}$. (Here $w(e)$ is the weight of the edge e.)

(Case ii) Suppose that $\pi(u,x) \cap D \neq \emptyset$ and $\pi(y,v) \cap D \neq \emptyset$. If $\pi(x,v) \cap D = \emptyset$, then y cannot appear on $\pi_{G-D}(x,v)$, which means the edge e is invalid. Otherwise we can safely add x into H.

(Case iii) Suppose that $\pi(u,x) \cap D = \emptyset$ but $\pi(y,v) \cap D \neq \emptyset$.

(Case iii.a) Suppose that (x,y) is not a tree edge in T_u. If $\pi(u,y) \cap D \neq \emptyset$, then we can safely add y into H. Otherwise, a similar argument as (Case iv) in Section 4.2.2 shows that we can discard e.

(Case iii.b) Suppose that (x,y) is a tree edge in T_u. Since x appears before y on $\pi_{G-D}(u,v)$, x has to be the parent of y in T_u (otherwise we discard (x,y)). If $T_u(y) \cap V(D) = \emptyset$, we add y into Helper; otherwise we discard y.

It is easy to see that if we add y into Helper, then y is u-clean. Now we need to show that whenever we discard y, $\pi_{G-D}(u,v)$ cannot go through the edge (x,y) (in the order of first x and then y). This is essentially the same as (Case vi) in Section 4.2.2. Note that since $T_u(y) \cap V(D) \neq \emptyset$, y lies on the tree T^u_{induced}. Since $\pi(u,y) \cap D = \emptyset$, if $\pi_{G-D}(u,v)$ goes through y, then $\pi(u,y)$ must be a prefix of $\pi_{G-D}(u,v)$. It follows that e_Δ is either equal to e or in the subtree $T_u(y)$. By (γ), e is not on the path $\pi(u,c^u)$, therefore e_Δ cannot be on the path $\pi(p^u,c^u)$ either, a contradiction.

(Case iv) Suppose that $\pi(u,x) \cap D \neq \emptyset$ but $\pi(y,v) \cap D = \emptyset$. This case is symmetric to (Case iii), so we only provide a sketch. If (x,y) is not a tree edge in T_v, then we add x into H if $\pi(x,v) \cap D \neq \emptyset$ and discard e otherwise. If (x,y) is a tree edge in T_v, then (assuming y is the parent of x in T_v) we add x into Helper if $T_v(x) \cap V(D) = \emptyset$ and discard e otherwise.

Summary. In Case III, we first construct T^u_{induced}, T^v_{induced}, T^u_{key}, and T^v_{key} in $\tilde{O}(d)$ time. Then we enumerate an edge $(p^u,c^u) \in T^u_{\text{key}}$ and an edge $(p^v,c^v) \in T^v_{\text{key}}$. Let $\mathcal{D}_\star = \mathcal{D}[u,v,c^u,c^v,0,0]$, then for each edge $e \in \mathcal{D}_\star \setminus D$ and each of its two possible orientations, according to the above case analysis, we either discard e, add a vertex into H, or add a vertex into Helper.

After this procedure, we have that $|H| \leq O(d^3)$ and $|\text{Helper}| \leq O(d^3)$. In this way, we can reduce Case III to $O(d^3)$ instances of Case II. Note that an instance of Case II runs in $O(d^3)$ time, therefore an invocation of Case III runs in $O(d^6)$ time.

5 CONCLUSIONS AND OPEN PROBLEMS

In this paper, we presented the first exact distance oracle that tolerates d edge failures and has reasonable size and query time bounds. Our oracle has size $O(dn^4)$ and query time $d^{O(d)}$. However, our oracle still has some drawbacks:

(1) We think the biggest drawback of our oracle is its preprocessing time. Is there a faster preprocessing algorithm for our oracle? In particular, can we preprocess it in $O(n^c)$ time for some constant c independent of d?

(2) Can we maintain exact distances under d *vertex* failures? Our oracle relies heavily on Theorem 3.1 which only works for edge failures.[6]

(3) Can we improve the size of our oracle to (say) $\tilde{O}(dn^2)$? Currently our oracle is trivial when $d=3$ or $d=4$ and only non-trivial when $d>4$. Such an improvement would imply non-trivial solutions for all d.

ACKNOWLEDGMENTS

We thank Yong Gu and Tianyi Zhang for helpful discussions during the initial stage of this research. We are grateful to Yaowei Long and Lijie Chen for helpful comments on a draft version of this paper. We also thank the anonymous STOC reviewers for helpful comments.

REFERENCES

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

[2] Josh Alman and Virginia Vassilevska Williams. 2021. A Refined Laser Method and Faster Matrix Multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*. 522–539. https://doi.org/10.1137/1.9781611976465.32

[6] [12, Section 4.1] proved a version of Theorem 3.1 for *approximate* distances under vertex failures. As the authors observe in their Footnote 15, a version for exact distances under vertex failures is unlikely to exist.

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

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

[5] Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. 2016. Multiple-Edge-Fault-Tolerant Approximate Shortest-Path Trees. In *33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 47)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 18:1–18:14. https://doi.org/10.4230/LIPIcs.STACS.2016.18

[6] Shiri Chechik and Sarel Cohen. 2020. Distance sensitivity oracles with subcubic preprocessing time and fast query time. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*. ACM, 1375–1388. https://doi.org/10.1145/3357713.3384253

[7] Shiri Chechik, Sarel Cohen, Amos Fiat, and Haim Kaplan. 2017. $(1+\epsilon)$-Approximate f-Sensitive Distance Oracles. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*. 1479–1496. https://doi.org/10.1137/1.9781611974782.96

[8] Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. 2012. f-sensitivity distance oracles and routing schemes. *Algorithmica* 63, 4 (2012), 861–882. https://doi.org/10.1007/s00453-011-9543-0

[9] Don Coppersmith and Shmuel Winograd. 1990. Matrix Multiplication via Arithmetic Progressions. *J. Symb. Comput.* 9, 3 (1990), 251–280. https://doi.org/10.1016/S0747-7171(08)80013-2

[10] Camil Demetrescu and Giuseppe F. Italiano. 2004. A new approach to dynamic all pairs shortest paths. *J. ACM* 51, 6 (2004), 968–992. https://doi.org/10.1145/1039488.1039492

[11] Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. 2008. Oracles for Distances Avoiding a Failed Node or Link. *SIAM J. Comput.* 37, 5 (2008), 1299–1318. https://doi.org/10.1137/S0097539705429847

[12] Ran Duan, Yong Gu, and Hanlin Ren. 2021. Approximate Distance Oracles Subject to Multiple Vertex Failures. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*. SIAM, 2497–2516. https://doi.org/10.1137/1.9781611976465.148

[13] Ran Duan and Seth Pettie. 2009. Dual-failure distance and connectivity oracles. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*. SIAM, 506–515. https://doi.org/10.1137/1.9781611973068.56

[14] Ran Duan and Seth Pettie. 2010. Connectivity Oracles for Failure Prone Graphs. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing* (Cambridge, Massachusetts, USA) *(STOC '10)*. ACM, New York, NY, USA, 465–474. https://doi.org/10.1145/1806689.1806754

[15] Ran Duan and Seth Pettie. 2020. Connectivity Oracles for Graphs Subject to Vertex Failures. *SIAM J. Comput.* 49, 6 (2020), 1363–1396. https://doi.org/10.1137/17M1146610

[16] Ran Duan and Tianyi Zhang. 2017. Improved Distance Sensitivity Oracles via Tree Partitioning. In *Algorithms and Data Structures - 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10389)*. Springer, 349–360. https://doi.org/10.1007/978-3-319-62127-2_30

[17] François Le Gall. 2014. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC'14, Kobe, Japan, July 23-25, 2014*. ACM, 296–303. https://doi.org/10.1145/2608628.2608664

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

[19] Yong Gu and Hanlin Ren. 2021. Constructing a Distance Sensitivity Oracle in $O(n^{2.5794}M)$ Time. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference) (LIPIcs, Vol. 198)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 76:1–76:20. https://doi.org/10.4230/LIPIcs.ICALP.2021.76

[20] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. 1987. Matching is as easy as matrix inversion. *Comb.* 7, 1 (1987), 105–113. https://doi.org/10.1007/BF02579206

[21] Mihai Pătraşcu and Mikkel Thorup. 2007. Planning for Fast Connectivity Updates. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*. IEEE Computer Society, 263–271. https://doi.org/10.1109/FOCS.2007.59

[22] Hanlin Ren. 2022. Improved distance sensitivity oracles with subcubic preprocessing time. *J. Comput. Syst. Sci.* 123 (2022), 159–170. https://doi.org/10.1016/j.jcss.2021.08.005

[23] Andrew James Stothers. 2010. *On the complexity of matrix multiplication*. Ph. D. Dissertation. The University of Edinburgh.

[24] Noam Ta-Shma. 2015. A simple proof of the Isolation Lemma. *Electron. Colloquium Comput. Complex.* (2015). https://eccc.weizmann.ac.il/report/2015/080

[25] Jan van den Brand and Thatchaphol Saranurak. 2019. Sensitive Distance and Reachability Oracles for Large Batch Updates. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*. IEEE Computer Society, 424–435. https://doi.org/10.1109/FOCS.2019.00034

[26] Oren Weimann and Raphael Yuster. 2013. Replacement Paths and Distance Sensitivity Oracles via Fast Matrix Multiplication. *ACM Trans. Algorithms* 9, 2, Article 14 (March 2013), 13 pages. https://doi.org/10.1145/2438645.2438646

[27] Virginia Vassilevska Williams. 2012. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*. ACM, 887–898. https://doi.org/10.1145/2213977.2214056