Approximate Single-Source Fault Tolerant Shortest Path

SURENDER BASWANA, Department of CSE, I.I.T. Kanpur, India
KEERTI CHOUDHARY, Department of Computer Science, Tel Aviv University, Israel
MOAZZAM HUSSAIN, WorldQuant Research, India
LIAM RODITTY, Department of Computer Science, Bar Ilan University, Israel

Let $G = (V, E)$ be an n-vertices m-edges directed graph with edge weights in the range $[1, W]$ for some parameter W, and $s \in V$ be a designated source. In this article, we address several variants of the problem of maintaining the $(1 + \epsilon)$-approximate shortest path from s to each $v \in V \setminus \{s\}$ in the presence of a failure of an edge or a vertex.

From the *graph theory* perspective, we show that G has a subgraph H with $\widetilde{O}(\epsilon^{-1} n \log W)$ edges such that for any $x, v \in V$, the graph $H \setminus x$ contains a path whose length is a $(1 + \epsilon)$-approximation of the length of the shortest path from s to v in $G \setminus x$. We show that the size of the subgraph H is optimal (up to logarithmic factors) by proving a lower bound of $\Omega(\epsilon^{-1} n \log W)$ edges. Demetrescu, Thorup, Chowdhury, and Ramachandran (SICOMP 2008) showed that the size of a fault tolerant exact shortest path subgraph in weighted directed/undirected graphs is $\Omega(m)$. Parter and Peleg (ESA 2013) showed that even in the restricted case of unweighted undirected graphs, the size of any subgraph for the exact shortest path is at least $\Omega(n^{1.5})$. Therefore, a $(1 + \epsilon)$-approximation is the best one can hope for.

We consider also the *data structure* problem and show that there exists an $\widetilde{O}(\epsilon^{-1} n \log W)$ size oracle that for any $v \in V$ reports a $(1 + \epsilon)$-approximate distance of v from s on a failure of any $x \in V$ in $O(\log \log_{1+\epsilon}(nW))$ time. We show that the size of the oracle is optimal (up to logarithmic factors) by proving a lower bound of $\Omega(\epsilon^{-1} n \log W \log^{-1} n)$. Finally, we present two *distributed algorithms*. We present a single-source *routing scheme* that can route on a $(1 + \epsilon)$-approximation of the shortest path from a fixed source s to any destination t in the presence of a fault. Each vertex has a label and a routing table of $\widetilde{O}(\epsilon^{-1} \log W)$ bits. We present also a *labeling scheme* that assigns each vertex a label of $\widetilde{O}(\epsilon^{-1} \log W)$ bits. For any two vertices $x, v \in V$, the labeling scheme outputs a $(1 + \epsilon)$-approximation of the distance from s to v in $G \setminus x$ using *only* the labels of x and v.

CCS Concepts: • **Theory of computation → Graph algorithms analysis**; *Approximation algorithms analysis*;

Additional Key Words and Phrases: Fault-tolerant, subgraph, approximate-distances, oracle, routing

A preliminary version of this article appeared in the Proceedings of SODA 2018.
This work was partially supported by Israel Science Foundation (ISF) and University Grants Commission (UGC) of India.
Authors' addresses: S. Baswana, Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, Kanpur 208016, Uttar Pradesh, India; email: sbaswana@cse.iitk.ac.in; K. Choudhary, Department of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel; email: keerti.india@gmail.com; M. Hussain, Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, Kanpur 208016, Uttar Pradesh, India; email: moazzamh94@gmail.com; L. Roditty, Department of Computer Science, Bar Ilan University, Ramat-Gan 52900, Israel; email: liam.roditty@biu.ac.il.

1549-6325/2020/07-ART44 $15.00
https://doi.org/10.1145/3397532

ACM Reference format:
Surender Baswana, Keerti Choudhary, Moazzam Hussain, and Liam Roditty. 2020. Approximate Single-Source Fault Tolerant Shortest Path. *ACM Trans. Algorithms* 16, 4, Article 44 (July 2020), 22 pages.
https://doi.org/10.1145/3397532

1 INTRODUCTION

In this article, we address several computational aspects of the problem of maintaining single-source approximate shortest paths for weighted directed graphs in presence of a failure. Let s be any designated source vertex in graph $G = (V, E)$ and ϵ be any positive fraction. We first consider the problem of computing a sparse subgraph H of G that for any $x, v \in V$ contains a $(1+\epsilon)$-multiplicative approximation of the shortest path from s to v in $G \setminus \{x\}$. Such a subgraph H is referred as a fault tolerant $(1+\epsilon)$-shortest path subgraph.

Demetrescu, Thorup, Chowdhury, and Ramachandran [14] showed that the size of a fault tolerant exact shortest path subgraph in weighted directed/undirected graphs is $\Omega(m)$; therefore, a $(1+\epsilon)$-approximation is the best that one can hope for. Parter and Peleg [23] showed that even in the case of unweighted undirected graphs, there are graphs such that the size of their fault tolerant exact shortest path subgraph is at least $\Omega(n^{1.5})$.

Bilò, Gualà, Leucci, and Proietti [8] showed that any weighted undirected graph has a fault tolerant $(1+\epsilon)$-shortest path subgraph of size $O((1/\epsilon^2)n \log n)$. Their result, however, uses techniques that rely heavily on the undirectedness of the graph and therefore cannot be extended to weighted directed graphs.

A fault tolerant $(1+\epsilon)$-shortest path subgraph for directed graphs is important both from theoretical and practical point of views. From the theoretical point of view, it is always an intriguing challenge to match or almost match in directed graphs the bounds that are known for undirected graphs. Finding small size subgraphs for directed graphs that preserve distance related properties of the graph is a hard task since many of the standard tools that are being used in undirected graphs rely on the fact that distances in undirected graphs are symmetric, which is obviously not the case in directed graphs.

Graph spanners are probably the most notable example for a separation between undirected and directed graphs. For every weighted undirected graph, there is a subgraph with $O(n^{1+1/k})$ edges that approximates the distances with a multiplicative stretch of $2k-1$ [24]. It is very easy to see that such a general result is not possible for directed graphs since no subgraph can approximate all distances of a complete bipartite graph.

The second problem that we address is designing a single-source compact routing scheme that after any failure is able to route on paths that are stretched by a factor of at most $(1+\epsilon)$. From the practical point of view, there are network routing scenarios in which routers only need to know which outgoing link to choose to get on a shortest path to a packet destination. More specifically, the Autonomous System's link-state database that is used by the Open Shortest Path First (OSPF) TCP/IP internet routing protocol is a directed graph [20]. Therefore, designing a single-source routing scheme that is resilient to link failures in directed graphs is of real practical need.

Lastly, we present an efficient oracle for reporting $(1+\epsilon)$-approximate distances from a source vertex after an occurrence of a failure. We also present a distributed implementation of this oracle (a labeling scheme).

Next, we formally state our results. All our results hold for the general case of weighted directed graphs where edge weights are assumed to be in the range $[1, W]$.

(1) Sparse subgraph: We show that it is possible to compute a subgraph $H \subseteq G$ with $O(n \log^3(n) \log_{1+\epsilon}(nW))$ edges such that for every $v, x \in V$, the distance of v from s in $H \setminus \{x\}$ is at most $(1+\epsilon)$

times the distance from s in $G \setminus \{x\}$. Moreover, the in-degree of each vertex in H is bounded by $O(\log^4(n) \log_{1+\epsilon}(nW))$. We also establish a lower bound of $\Omega(n \log_{1+\epsilon} W)$ on the size of such a subgraph.

(2) Oracle: We can build a data structure of size $O(n \log_{1+\epsilon}(nW))$ that can report in $O(\log \log_{1+\epsilon}(nW))$ time, for every $x, v \in V$, a $(1+\epsilon)$-approximate distance of v from s in $G\setminus\{x\}$. Also, we show that the size of the oracle is optimal (up to logarithmic factors) by proving a lower bound of $\Omega(n \log_{1+\epsilon}(W)/\log n)$. Our lower bound is independent of the time needed for reporting the $(1+\epsilon)$-approximate distances.

(3) Labeling scheme: Using this data structure, we can also get a very compact labeling scheme for reporting approximate distances from s under any vertex failure. Each vertex will store a label of $O(\log_{1+\epsilon}(nW) \log n)$ bits such that for any failing vertex x and a destination vertex v, it is possible to determine the $(1+\epsilon)$-approximate distance of v from s in $G\setminus\{x\}$ by processing the labels associated with v and x only.

(4) Routing: We provide a single-source routing scheme that can route on a $(1+\epsilon)$-approximate shortest path from source s to any destination $v \in V$ in the presence of a fault. Each vertex has a label of $O(\log^4(n) \log_{1+\epsilon}(nW))$ bits and a routing table of $O(\log^5 n \log_{1+\epsilon}(nW))$ bits. When the routing is started at the source, the labels of the faulty vertex and the destination vertex are assumed to be known.

It must be noted that we describe all our constructions with respect to vertex failure only. Edge failure can be handled by inserting a vertex, say z_{uv}, in the middle of each tree edge (u, v) on the shortest path tree rooted at source s. So the deletion of tree edge (u, v) is equivalent to the deletion of vertex z_{uv}.

1.1 Related Work

For undirected graphs, Baswana and Khanna [2] showed a vertex-fault tolerant 3-shortest path subgraph with $O(n \log n)$ edges. Bilò, Gualà, Leucci, and Proietti [8] showed that any weighted undirected graph has a vertex fault tolerant $(1+\epsilon)$-shortest path subgraph of size $O((1/\epsilon^2)n \log n)$. The same article also shows that the work of Nardelli et al. [21] can be used to compute an edge-fault tolerant 3-shortest path subgraph with at most $2n$ edges. Parter and Peleg [23] showed that any unweighted (un)directed graph has a fault tolerant exact shortest path subgraph with $O(n^{3/2})$ edges. They also showed a matching lower bound. Parter [22] extended this result to two edge failures with $O(n^{5/3})$ edges but only for undirected graphs. She also showed a lower bound of $\Omega(n^{5/3})$. Gupta and Khan [18] extended the work of Ref. [22] to vertex failures and directed graphs.

For the case of multiple edge faults, Bilò et al. [7] obtained a k-fault tolerant $(2k+1)$-shortest path subgraph with $O(kn)$ edges again only for undirected graphs. They also showed that there is a data-structure of size $O(kn \log^2 n)$ that reports the $(2k+1)$-approximate distance from s in $O(k^2 \log^2 n)$ time. Recently, Gupta and Singh [19] showed that any undirected graph can be preprocessed to obtain a 1-edge fault-tolerant data-structure of size $\tilde{O}(n^{3/2})$[1] that reports the exact distance from s in $\tilde{O}(1)$ time. For preserving exact distances in (un)directed graphs under multiple failures, Bodwin et al. [9] obtain a construction of a f-fault tolerant subgraph with $\tilde{O}(fn^{2-1/2^f})$ edges.

In the case of all-pair shortest paths (APSP), Demetrescu, Thorup, Chowdhury, and Ramachandran [14] showed that we can build an $O(n^2 \log n)$ size data structure that can report the distance from u to v avoiding x for any $u, v, x \in V$ in $O(1)$ time. The construction time of their data structure is $O(mn^2 + n^3 \log n)$. Bernstein and Karger further improved the preprocessing time

[1] $\tilde{O}()$ hides the poly-logarithmic factors.

to $\tilde{O}(\sqrt{m}n^2)$ [3] and finally to $\tilde{O}(mn)$ [4]. The latter preprocessing time matches, up to poly-logarithmic factors, the best-known runtime for the APSP in the same setting. Duan and Pettie [16, 17] extended the result of Ref. [14] to dual failures by designing a data structure of $O(n^2 \log^3 n)$ space that can answer any distance query upon the failure of two vertices in $O(\log n)$ time. The authors of Ref. [17] comment that their techniques do not seem to be extensible beyond two failures, and even an oracle for three failures seems too hard to achieve.

The questions of finding graph spanners, approximate and exact distance oracles, and compact routing schemes that are resilient to vertex or edge failures have also been studied in Refs [5], [6], [10], [11], [12], [15], and [26].

1.2 Organization of the Article

We describe notation and terminology in Section 2. In Section 3, we provide the main ideas used in our article. We present a randomized algorithm for computing a fault tolerant $(1+\epsilon)$-shortest path subgraph in Section 4. In order to use it for routing, the subgraph should satisfy some additional property. We present a deterministic algorithm for computing the subgraph with this additional property in Section 5. As a by-product, we are also able to bound the in-degree of the vertices in the subgraph. The oracle and the labeling scheme are described in Section 6, and the routing scheme is described in Section 7. In Section 8, we present our lower bounds.

2 PRELIMINARIES

Let $G = (V, E)$ be a directed graph on $n = |V|$ vertices and $m = |E|$ edges, and $s \in V$ be the designated source vertex. We assume that the weight of each edge is a real number greater than one and bounded above by some threshold, say W. Below, we introduce some of the notation that will be used throughout the article.

- $wt(u, v)$: Weight of edge (u, v) in G.
- $wt(P)$: Weight of path P in G, that is, if P is $(u_0, \ldots, u_t)$, then $wt(P) = \sum_{i=0}^{t-1} wt(u_i, u_{i+1})$.
- T: A shortest path tree of G rooted at s.
- $T(v)$: Subtree of T rooted at a vertex v.
- $parent_T(v)$: Parent of vertex v in T.
- $\textsc{path}_T(a, b)$: Path from vertex a to vertex b in T.
- $\textsc{path}_T(\bar{a}, b)$: $\textsc{path}_T(a, b) \setminus \{a\}$.
- $\textsc{path}_T(a, \bar{b})$: $\textsc{path}_T(a, b) \setminus \{b\}$.
- $depth_T(v)$: Depth of vertex v in T.
- $P[\cdot, b]$: The prefix of path P up to vertex b, assuming that b lies in P.
- $P[a, b]$: The sub-path of path P lying between vertices a, b, assuming a precedes b on P.
- $\sigma(P)$: Subsequence of those vertices of path P whose incoming edge in the path is a non-tree edge.
- $\textsc{freq}(w, C)$: The number of sequences of set C in which vertex w appears.
- $P::Q$: The path formed by concatenating paths P and Q in G. Here, it is assumed that the last vertex of P is the same as the first vertex of Q.
- $dist_H(u, v)$: Distance of v from u in graph H.
- $G \setminus x$: Graph obtained by deleting node x from G.
- $\textsc{powers}(c)$: Set of all powers of c in range $[1, nW]$.

We start by defining a detour:

Definition 2.1. A simple path $P = (u_0, \ldots, u_t = v)$ in G is said to be a detour to v with respect to T if u_0 is an ancestor of v, and for $0 < i < t$, none of the u_i's is an ancestor of v in T.

Notice that it follows from the above definition that tree edges and forward edges are also detours of size one. Next, we define a special class of detours, the tree-path favoring detours, which will be used in our constructions.

Definition 2.2. A detour D from u to v is a tree-path favoring detour if for any $a, b \in D \setminus \{u, v\}$, where a precedes b in D and a is an ancestor of b in T, it holds that the segment $D[a, b]$ is a tree path.

It is easy to see that if a detour is not a tree-path favoring detour, then we can easily convert it into a tree-path favoring detour, by repeatedly replacing segment $D(a, b)$ with the tree path $\text{PATH}_T(a, b)$. Since T is the shortest path tree, by doing this, we do not increase the weight of the detour. So, henceforth, we can assume that all the detours referred to in this article are tree-path favoring detours.

For the sake of simplicity, we use the following alternative weight function (usually known as the Johnson transformation), which is quite popular in the literature of shortest paths.

$$wt^*(u, v) := wt(u, v) + dist_G(s, u) - dist_G(s, v)$$

It is easy to see that for any edge (u, v), $wt^*(u, v)$ is non-negative. Also, if (u, v) is a tree edge, then $wt^*(u, v)$ must be zero. From the next lemma, it follows that for any detour D, $wt^*(D)$ must be bounded by nW.

Lemma 2.1. *Let $a, b \in T$ be such that a is an ancestor of b, and let P be any path from a to b. Then, $wt^*(P) = wt(P) - dist_G(a, b)$.*

Proof. Let P be equal to $(a = a_0, \ldots, a_t = b)$. So,

$$\begin{aligned} wt^*(P) &= \textstyle\sum_{i=1}^{t} wt^*(a_{i-1}, a_i) \\ &= \textstyle\sum_{i=1}^{t} (wt(a_{i-1}, a_i) + dist_G(s, a_{i-1}) - dist_G(s, a_i)) \\ &= dist_G(s, a_0) - dist_G(s, a_t) + \textstyle\sum_{i=1}^{t} wt(a_{i-1}, a_i) \\ &= wt(P) + dist_G(s, a) - dist_G(s, b) \end{aligned}$$

Thus, we get $wt^*(P) = wt(P) - dist_G(a, b)$. □

In the next Lemma, we show an important property of certain detours, which we use for constructing our subgraph. This Lemma uses the new weight function and exemplifies its significance.

Lemma 2.2. *Let $x, v \in V$ be such that x is an ancestor of v in T. There is a shortest path from s to v in $G \setminus x$ of the form - $\text{PATH}_T(s, a)$::D::$\text{PATH}_T(b, v)$, where D is a detour starting from a vertex $a \in \text{PATH}_T(s, \bar{x})$ and terminating at a vertex $b \in \text{PATH}_T(\bar{x}, v)$ for which $wt^*(D)$ is minimum.*

Proof. Let P be some shortest path from s to v in $G \setminus x$. Let a be the last vertex of P that is also in $\text{PATH}_T(s, \bar{x})$, and let b be the first successor of a in P that is in $\text{PATH}_T(\bar{x}, v)$. It is easy to see that such two vertices a, b must exist. By the detour definition, it also follows that $P[a, b]$ is a detour for b, as none of its internal vertices can be an ancestor of b. Thus, we can set D to be $P[a, b]$. Since T is the shortest path tree, we can replace the segments $P[s, a]$ and $P[b, v]$ with $\text{PATH}_T(s, a)$ and $\text{PATH}_T(b, v)$, respectively, without increasing the total weight. Thus, the path $Q = \text{PATH}_T(s, a)$::D::$\text{PATH}_T(b, v)$ forms a shortest path from s to v in $G \setminus x$. Note that $wt(Q) = dist_G(s, v) + (wt(D) - dist_G(a, b))$, which equals $dist_G(s, v) + wt^*(D)$. For every other path Q' of the form $\text{PATH}_T(s, a')$::D'::$\text{PATH}_T(b', v)$, where $a' \in \text{PATH}_T(s, \bar{x})$ and $b' \in \text{PATH}_T(\bar{x}, v)$, it holds that $wt(Q') = dist_G(s, v) + wt^*(D')$. Now, since $wt(Q) = wt(P)$ and P is a shortest path, it follows that D must be a detour with minimum wt^* value among all detours, which starts at a vertex in $\text{PATH}_T(s, \bar{x})$ and terminates at a vertex in $\text{PATH}_T(\bar{x}, v)$. □

We now state a simple lemma that will be used to obtain a randomized construction for a sparse subgraph.

Lemma 2.3. *Let C be a collection of at most n^2 subsets of V, each of size exactly $4c\ln(n)$. If we pick a subset S of V of size n/c uniformly at random, then, with probability at least $1 - 1/n^2$, for each set $W \in C$, $W \cap S$ is non-empty.*

Proof. Let $t = (4c \times \ln(n))$. Note that there are a total of $\binom{n}{n/c}$ possibilities for set S. Now for any subset $A \in C$, the probability that $A \cap S$ is empty is

$$\frac{\binom{n-t}{n/c}}{\binom{n}{n/c}} \leq \left(\frac{n-t}{n}\right)^{n/c} = \left(1 - \frac{1}{n/t}\right)^{n/c} = \left(1 - \frac{4\ln(n)}{n/c}\right)^{n/c},$$

which is at most $1/n^4$. On applying the union bound, we get that the probability that there exists a $W \in C$ for which $W \cap S$ is non-empty is at most $1/n^2$. □

3 MAIN IDEAS

For any $\alpha > 0$, let $\mathrm{HD}_\alpha(v)$ be a detour that originates from the highest possible ancestor of v in T and terminates at v such that its weight $wt^*(\mathrm{HD}_\alpha(v))$ is less than or equal to α, that is, there is no other detour that ends at v and starts at a higher ancestor of v whose weight is bounded by α. We denote the first vertex of $\mathrm{HD}_\alpha(v)$ with $\textsc{first}_\alpha(v)$. Consider the following subgraph:

$$H = T \bigcup \left(\bigcup_{\substack{b \in V \\ \alpha \in \textsc{powers}(1+\epsilon)}} \mathrm{HD}_\alpha(b) \right)$$

For any $x, v \in V$, the graph $H \setminus x$ contains a $(1+\epsilon)$-approximation of the shortest path from s to v in $G \setminus x$. This is because if the shortest path takes a detour D from a to b to avoid x, then H will contain a detour D' that starts at a or one of its ancestors, ends at b, and $wt^*(D') \leq (1+\epsilon)wt^*(D)$. Based on this key observation, we are able to compute an oracle of size $O(n\log_{1+\epsilon}(nW))$ for reporting $(1+\epsilon)$-approximate distance from the source upon failure of any vertex in $O(\log\log_{1+\epsilon}(nW))$ time. Though the subgraph H can be seen as a fault tolerant $(1+\epsilon)$-shortest path subgraph, its size can be as large as $\Theta(m)$. This is because even a single detour may contain n edges. So storing $\mathrm{HD}_\alpha(v)$ for each v and each α may require $\Omega(m)$ space in the worst case. In order to achieve sparseness for H, our starting point is the sub-structure property of $\mathrm{HD}_\alpha(v)$ stated in the following lemma.

Lemma 3.1. *Let $D = HD_\alpha(v)$ be a detour for v, for some $\alpha > 0$. Then, for any vertex $w \in \sigma(D)$, the segment $D[\cdot, w]$ is also a detour.*

Proof. Let u be the first vertex on D, that is, $u = \textsc{first}_\alpha(v)$. We first show that all the vertices of D must lie in the subtree $T(u)$. Assume this is not the case, and let y be the last vertex of D that is not in the subtree $T(u)$. Also, let z be the Lowest Common Ancestor (LCA) of y and u. Consider the path $D' = \textsc{path}_T(z, y)::D[y, v]$. It is easy to see that D' is a detour for v. Also, the set of non-tree edges of D' is a subset of the non-tree edges of D, thus $wt^*(D') \leq wt^*(D) \leq \alpha$. But this contradicts the definition of D, as D' is a detour for v starting from an ancestor of u with wt^* at most α.

From the above discussion, it follows that u must be an ancestor of w. Therefore, it suffices to show now that none of the internal vertices of $D[u, w]$ are ancestors of w. Let us suppose there exists a vertex $z \in D[u, w]$ $(z \neq u, w)$ such that z is an ancestor of w. But in such a case, we can replace the segment $D[z, w]$ of detour D with $\textsc{path}_T(z, w)$, thereby contradicting the fact that D is a tree path favoring detour. Hence, $D[u, w]$ cannot pass through any ancestor of w, except for the starting vertex u. □

It follows from Lemma 3.1 that for any $w \in \sigma(\text{HD}_\alpha(v))$, if H contains a $(1+\epsilon)$-approximation of the detour $\text{HD}_\alpha(v)[\cdot, w]$, then on including just the suffix $\text{HD}_\alpha(v)[w, v]$, we can see that H will contain a $(1+\epsilon)$-approximation of $\text{HD}_\alpha(v)$. A simple way to include a $(1+\epsilon)$-approximation of the detour $\text{HD}_\alpha(v)[\cdot, w]$ would be to add to H the detours $\text{HD}_\beta(w)$, for each $\beta \in \text{POWERS}(1+\epsilon)$. However, to get a sparse subgraph instead of including each $\text{HD}_\beta(w)$, we can use the same trick recursively and include a further $(1+\epsilon)$-approximation of $\text{HD}_\beta(w)$. This motivates for a hierarchical computation of H in some $k(\geq 2)$ rounds, where in a round, we only add a short suffix of $\text{HD}_\alpha(v)$ to the subgraph H, and we move its prefix (which is a detour in itself) to the next round to be processed recursively. This constitutes the main idea for constructing a sparse subgraph and a compact routing scheme for approximate shortest paths from s under failure of any vertex.

4 SPARSE SUBGRAPH

For the sake of better exposition of the algorithm, we first present the construction of a subgraph with $\widetilde{O}(n^{1.5}\log_{1+\epsilon}(nW))$ edges using a hierarchy of two levels. In the next section, we extend it to a k-level hierarchical construction that achieves a bound of $\widetilde{O}(n\log_{1+\epsilon}(nW))$ on the size of the subgraph.

4.1 Sparse Subgraph with $\widetilde{O}(n^{1.5}\log_{1+\epsilon}(nW))$ Edges

In this section, we give a construction of a sparse subgraph H with $\widetilde{O}(n^{1.5}\log_{1+\epsilon}(nW))$ edges that with high probability[2] preserves the approximate shortest paths from s after a failure of any vertex. The underlying idea used in the construction of this subgraph is the following. We pick a small set S of vertices uniformly and at random. For each $v \in S$ and for each $\alpha \in \text{POWERS}(1+\epsilon)$, we include $\text{HD}_\alpha(v)$ in the subgraph H. We cannot afford to include $\text{HD}_\alpha(u)$ for every $u \in V \setminus S$, so we include only a *small* suffix of $\text{HD}_\alpha(u)$. Due to the random sampling used to construct S, it turns out that if $\text{HD}_\alpha(u)$ is long, then its small suffix will have a vertex, say w, from S. The detour to w concatenated with the small suffix of $\text{HD}_\alpha(u)$ will preserve $\text{HD}_\alpha(u)$ approximately. In Algorithm 4.1, we present the pseudocode for computing a subgraph H with $\widetilde{O}(n^{1.5}\log_{1+\epsilon}(nW))$ edges that is based on this idea.

ALGORITHM 4.1: Computation of subgraph with $\widetilde{O}(n^{1.5}\log_{1+\epsilon}(nW))$ edges.

```
1 H ← T;
2 foreach v ∈ V and α ∈ POWERS(1 + ε) do
3 |   Add last √n non-tree edges of HD_α(v) to H;
4 S ← A uniformly random set of 4√n log_e n vertices;
5 foreach v ∈ S and α ∈ POWERS(1 + ε) do
6 |   Add all non-tree edges of HD_α(v) to H;
```

We first show that with high probability any long detour ($> \sqrt{n}$) has in its $\sqrt{n}$ length suffix at least one vertex from set S.

LEMMA 4.1. *With high probability, the following holds—For each $\alpha \in \text{POWERS}(1+\epsilon)$ and each $v \in V$, if $HD_\alpha(v)$ contains more than $\sqrt{n}$ non-tree edges, then the set of last $\sqrt{n}$ vertices of $\sigma(HD_\alpha(v))$ contain a vertex from set S.*

[2]Throughout the article, we use the term "with high probability (w.h.p.)" to denote that the probability of the respective event is at least $1 - 1/n^2$.

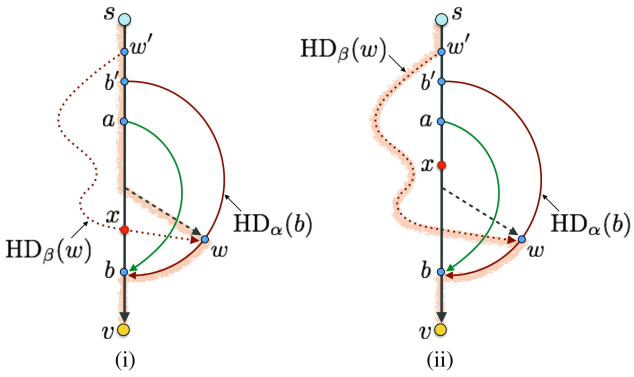

Fig. 1. Approximate shortest path from s to v in $H \setminus x$ in the subcases: (i) x is not an ancestor of w in T, and (ii) x is an ancestor of w.

Proof. Consider the collection C defined below.

$$C = \{\text{last } \sqrt{n} \text{ vertices of } \sigma(\text{HD}_\alpha(v)) \mid v \in V,$$
$$\alpha \in \text{powers}(1+\epsilon), |\sigma(\text{HD}_\alpha(v))| > \sqrt{n}\}$$

The collection C will contain at most n^2 sets (each of size $\sqrt{n}$), since for any vertex v, we can have at most n detours corresponding to n ancestors of v in T. So, the result follows by simply applying Lemma 2.3. □

We will now show that upon a failure of any vertex x, the shortest paths from s in graph $G \setminus x$ are stretched by a factor of at most $(1+\epsilon)^2$ in $H \setminus x$.

Lemma 4.2. *For every two vertices $x, v \in V$, w.h.p.,*

$$dist_{H\setminus x}(s,v) \leq (1+\epsilon)^2 dist_{G\setminus x}(s,v).$$

Proof. Let P be the shortest path from s to v in $G \setminus x$. If x is not an ancestor of v in T, then P will be just the tree path from s to v in T. Thus, let us consider the case when x is an ancestor of v. By Lemma 2.2, the path P can be represented as $\text{path}_T(s,a)::D::\text{path}_T(b,v)$, where D is a detour avoiding x. We take α to be the smallest power of $(1+\epsilon)$ greater than $wt^*(D)$. Let us consider the following two cases separately.

(1) **Case 1 :** $\text{HD}_\alpha(b)$ contains at most $\sqrt{n}$ non-tree edges.
In this case, $\text{HD}_\alpha(b)$ will lie in subgraph H. Since $\alpha \geq wt^*(D)$, $\text{first}_\alpha(b)$ must be either equal to a or an ancestor of a. So, instead of detour D, we can just follow the detour $\text{HD}_\alpha(b)$. Thus, the concatenation $Q = \text{path}_T(s, \text{first}_\alpha(b))::\text{HD}_\alpha(b)::\text{path}_T(b,v)$ forms a path from s to v in $H \setminus x$. Also, $wt^*(Q)$ is at most $(1+\epsilon)wt^*(P)$, as under the weight function wt^*, tree edges get zero weight.

(2) **Case 2 :** $\text{HD}_\alpha(b)$ contains more than $\sqrt{n}$ non-tree edges.
In this case, Lemma 4.1 implies that w.h.p. the last $\sqrt{n}$ vertices of $\sigma(\text{HD}_\alpha(b))$ must contain a vertex, say w, from set S. So the segment $\text{HD}_\alpha(b)[w,b]$ will lie in H because the last $\sqrt{n}$ non-tree edges of $\text{HD}_\alpha(b)$ are included in H. Also, by Lemma 3.1, the prefix $\text{HD}_\alpha(b)[\cdot,w]$ is a detour for vertex w. We further consider the following subcases. (See Figure 1).
(i) If x is not an ancestor of w, then simply take Q as $\text{path}_T(s,w)::\text{HD}_\alpha(b)[w,b]::\text{path}_T(b,v)$. Since $wt^*(\text{HD}_\alpha(b)[w,b]) \leq wt^*(\text{HD}_\alpha(b)) \leq (1+\epsilon)wt^*(D)$, we get that $wt^*(Q) \leq (1+\epsilon) \times wt^*(P)$.

(ii) We now consider the subcase when x is an ancestor of w in T. Notice that prefix $\text{HD}_\alpha(b)[\cdot, w]$ is not included in H, but we can take a further $(1+\epsilon)$-approximation of it. Let β be the smallest power of $(1+\epsilon)$ greater than $wt^*(\text{HD}_\alpha(b)[\cdot, w])$. Since $w \in S$, $\text{HD}_\beta(w)$ will lie in subgraph H. So, we take Q as $\text{PATH}_T(s, \text{FIRST}_\beta(w))::\text{HD}_\beta(w)::\text{HD}_\alpha(b)[w, b]$ concatenated with $\text{PATH}_T(b, v)$. Notice that $\text{HD}_\beta(w)$ cannot contain x, since $\text{FIRST}_\beta(w)$ is either the same as or an ancestor of the first vertex of $\text{HD}_\alpha(b)[\cdot, w]$. Finally, $wt^*(\text{HD}_\beta(w)::\text{HD}_\alpha(b)[w, b])$ is bounded by $(1+\epsilon)wt^*(\text{HD}_\alpha(b)) \leq (1+\epsilon)^2 \times wt^*(D)$. Hence, $wt^*(Q)$ is at most $(1+\epsilon)^2 \times wt^*(P)$.

In all the above cases/subcases, we were able to show that w.h.p. there exists a path Q such that $wt^*(Q) \leq (1+\epsilon)^2 \times wt^*(P)$. Now, as s is an ancestor of v, on adding $dist_G(s, v)$ on both sides and applying Lemma 2.1, we get that $wt(Q) \leq (1+\epsilon)^2 \times wt(P)$. □

We conclude with the following theorem.

Theorem 4.3. *Let G be a directed weighted graph on n vertices with maximum edge weight W and s be the designated source vertex. Then, we can compute in polynomial time a subgraph H with $O(n^{1.5}\log(n)\log_{1+\epsilon}(nW))$ edges such that with high probability, the following relation holds:*

$$\text{For any } x, v \in V, dist_{H\setminus x}(s, v) \leq (1+\epsilon)^2 dist_{G\setminus x}(s, v).$$

4.2 Sparse Subgraph with $\widetilde{O}(n\log_{1+\epsilon}(nW))$ Edges

In the $\widetilde{O}(n^{1.5}\log_{1+\epsilon}(nW))$ size subgraph described in the previous section, we constructed a 2-level hierarchy of vertices, namely, S and V. The detour to vertices in set S and the short suffixes of detours of vertices in V could preserve every detour D up to a stretch of $(1+\epsilon)^2$. In order to further improve the size of the subgraph, we form a finer hierarchy of subsets of vertices: $S_1, \ldots, S_k$ for some $k > 2$. As we go up in this hierarchy, the size of these sets decreases and, thus, we can afford to store *longer* suffixes of detours from their vertices. In particular, for a given $i \in [1, k]$, S_i will have at most $n^{1-(i-1)/k}$ vertices and from each vertex $v \in S_i$, we store suffixes of $\widetilde{O}(n^{i/k}\log_{1+\epsilon}(nW))$ length. Similar to the 2-level case, a combination of k types of these detour suffixes will preserve every detour D up to a factor $(1+\epsilon)^k$. We now formalize this key idea by defining a $(1+\epsilon, t)$-preserver of a detour as follows.

Definition 4.1. Let D be a detour from u to v, $\epsilon \in (0, 1)$ be some real number, and t be some positive integer. Also let α be the smallest power of $(1+\epsilon)$ greater than or equal to $wt^*(D)$. Then, a $(1+\epsilon, t)$-preserver of D is a path in G obtained as follows. (See Figure 2).

(1) If $t = 1$, then a $(1+\epsilon, 1)$-preserver of D is just $\text{HD}_\alpha(v)$.
(2) If $t > 1$, then it is obtained by concatenating (i) a $(1+\epsilon, t-1)$-preserver of prefix $\text{HD}_\alpha(v)[\cdot, w]$, and (ii) the suffix $\text{HD}_\alpha(v)[w, v]$, for some vertex $w \in \sigma(\text{HD}_\alpha(v))$.

Remark 4.1. For Definition 4.1 to be well defined, we require that $\text{HD}_\alpha(v)[\cdot, w]$ is a detour for vertex w. This is indeed true since $w \in \sigma(\text{HD}_\alpha(v))$, and the proof follows from Lemma 3.1.

The following lemma shows the significance of a $(1+\epsilon, t)$-preserver. The proof of the lemma is similar to proof of Lemma 4.2.

Lemma 4.4. *Let D be a detour from a to b, and H be a subgraph of G containing tree T and a $(1+\epsilon, t)$-preserver for D. Then, for any internal vertex x lying on $\text{PATH}_T(a, b)$, the graph $H \setminus x$ contains a path, say Q, from s to b whose weight (i.e., $wt(Q)$) is at most $(1+\epsilon)^t$ times $wt(\text{PATH}_T(s, u)::D)$.*

Proof. In order to prove the lemma, it suffices to show that the following claim holds.

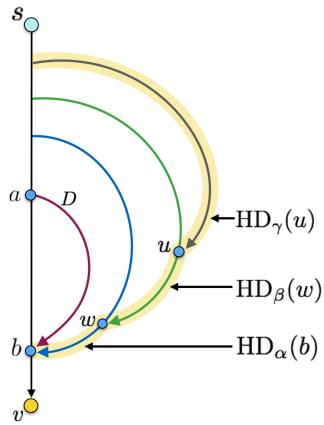

Fig. 2. The path highlighted in yellow color depicts a $(1+\epsilon, 3)$-preserver of detour D. Here, α, β, γ are, respectively, the smallest powers of $(1+\epsilon)$, greater than or equal to the weight of (i) D, (ii) $\text{HD}_\alpha(b)[\cdot, w]$, and (iii) $\text{HD}_\beta(w)[\cdot, u]$.

Claim: Let D be a detour from a to b, and H be a subgraph of G containing tree T and a $(1+\epsilon, t)$-preserver for D. Then, for any internal vertex x lying on $\text{PATH}_T(a, b)$, the graph $H \setminus x$ contains a path, say Q, from s to b such that $wt^*(Q)$ is at most $(1+\epsilon)^t \times wt^*(\text{PATH}_T(s, u)::D)$.

We prove the claim by applying induction on integer t. Let us denote by P the path $\text{PATH}_T(s, u)::D$, then $wt^*(P) = wt^*(D)$. Now, let α be the smallest power of $(1+\epsilon)$ greater than or equal to $wt^*(D)$. Since $\alpha \geq wt^*(D)$, the first vertex on $\text{HD}_\alpha(b)$, i.e., $\text{FIRST}_\alpha(b)$ must be either a or an ancestor of a. We first prove that the base case holds true, and later, using induction, give the proof for the generic case.

Base case. If $t = 1$, then $\text{HD}_\alpha(b)$ will lie in H. So path $Q = \text{PATH}_T(s, \text{FIRST}_\alpha(b))::\text{HD}_\alpha(b)$ forms a path from s to v in $H \setminus x$. Also $wt^*(Q)$ is at most $(1+\epsilon) \times wt^*(P)$.

Generic case. If $t > 1$, then there will exist a vertex $w \in \sigma(\text{HD}_\alpha(b))$ for which H contains both the suffix $\text{HD}_\alpha(b)[w, b]$ and a $(1+\epsilon, t-1)$-preserver of the prefix $\text{HD}_\alpha(b)[\cdot, w]$. We have the following two subcases depending upon whether or not x is an ancestor of w.

(i) Let us first consider the scenario when x is an ancestor of w. Though the prefix $\text{HD}_\alpha(b)[\cdot, w]$ does not lie in H, it can be seen that H contains a $(1+\epsilon, t-1)$-preserver of it. Since x is an internal vertex of $\text{PATH}_T(\text{FIRST}_\alpha(b), w)$, by induction hypothesis, $H \setminus x$ must contain a path Q_0 such that $wt^*(Q_0)$ is at most $(1+\epsilon)^{t-1} wt^*(\text{PATH}_T(s, \text{FIRST}_\alpha(b))::\text{HD}_\alpha(\text{b})[\cdot, \text{w}])$. We choose Q to be the path $Q_0::\text{HD}_\alpha(b)[w, b]$. It is easy to see that Q is a path from s to b in H avoiding x. Also,

$$\begin{aligned}
wt^*(Q) &= wt^*(Q_0) + wt^*(\text{HD}_\alpha(b)[w, b]) \\
&\leq (1+\epsilon)^{t-1} wt^*\Big(\text{PATH}_T(s, \text{FIRST}_\alpha(b))::\text{HD}_\alpha(\text{b})[\cdot, \text{w}]\Big) \\
&+ wt^*(\text{HD}_\alpha(b)[w, b]) \\
&\leq (1+\epsilon)^{t-1} wt^*\Big(\text{PATH}_T(s, \text{FIRST}_\alpha(b))::\text{HD}_\alpha(\text{b})\Big) \\
&\leq (1+\epsilon)^t wt^*(\text{PATH}_T(s, u) :: D)
\end{aligned}$$

Thus, in this subcase, we are able to show that $wt^*(Q)$ is at most $(1+\epsilon)^t wt^*(P)$.

(ii) If x is not an ancestor of w in T, then we can simply take Q to be $\text{PATH}_T(s, w)::\text{HD}_\alpha(b)[w, b]$. Notice that $wt^*(\text{HD}_\alpha(b)[w, b]) \leq wt^*(\text{HD}_\alpha(b))$, which is at most$(1+\epsilon) wt^*(D)$. Thus, in this subcase, $wt^*(Q)$ is at most $(1+\epsilon) wt^*(P) \leq (1+\epsilon)^t wt^*(P)$. This completes our proof. □

We now describe the algorithm for computing the sparse subgraph H. The algorithm consists of k rounds that operate on k sets, namely, $S_1, S_2, \ldots, S_k$ as follows. Let $S_1 = V$ be the initial set. In any round $i < k$, we first compute a uniformly random subset of V of size $O(n/n^{i/k})$, and move these vertices to S_{i+1}. Next, for each each $v \in S_i$ and each $\alpha \in \text{POWERS}(1+\epsilon)$,

(1) If $\sigma(\text{HD}_\alpha(v))$ does not contain any vertex from S_{i+1}, then the complete detour $\text{HD}_\alpha(v)$ is added to H.
(2) Otherwise, if z is the last vertex of $\sigma(\text{HD}_\alpha(v))$ that lies in S_{i+1}, then only the suffix $\text{HD}_\alpha(w)[z, w]$ is added to H.

Finally, in round k, for each each $v \in S_k$ and each $\alpha \in \text{POWERS}(1+\epsilon)$, we add $\text{HD}_\alpha(v)$ to H. Also, the shortest path tree T is added to the graph H.

Algorithm 4.2 presents the pseudocode for this construction. In the algorithm, we use the notation $\text{SUFFIX}(\sigma, A)$ to denote the maximal suffix of σ that does not contain an element of A, where σ is any sequence of vertices and A is a subset of V. Notice that, instead of Step 1 and Step 2 stated above, we can equivalently just say that for each $w \in \text{SUFFIX}(\sigma(\text{HD}_\alpha(v)), S_{i+1})$, the incoming edge of w in $\text{HD}_\alpha(v)$ is added to H.

ALGORITHM 4.2: Computation of subgraph with $\widetilde{O}(n \log_{1+\epsilon}(nW))$ edges.

```
1 H ← T;
2 S_1 ← V(G);
3 for i = 1 to k-1 do
4 |   S_{i+1} ← A uniformly random subset of V of size O(n/n^{i/k});
5 |   foreach v ∈ S_i and α ∈ POWERS(1+ε) do
6 |   |   foreach w ∈ SUFFIX(σ(HD_α(v)), S_{i+1}) do
7 |   |   |   Add incoming edge of w in HD_α(v) to H;
8 foreach v ∈ S_k and α ∈ POWERS(1+ε) do
9 |   Add HD_α(v) to H;
```

The following lemma shows that the subgraph H contains a $(1+\epsilon, k)$-preserver for each possible detour D in G.

LEMMA 4.5. *The graph H computed by Algorithm 4.2 contains a $(1+\epsilon, k)$-preserver for each possible detour D in G.*

PROOF. In order to prove the lemma, it suffices to show that the following claim holds.

Claim: Let i be any index in $[1, k]$. Then, for any $v \in S_{k-i+1}$, the graph H contains a $(1+\epsilon, i)$ preserver of each detour terminating at v.

We prove the above claim by applying induction on index i. To prove the base case, i.e., $i = 1$, consider any vertex $v \in S_k$. Since H contains $\text{HD}_\alpha(v)$ for each $\alpha \in \text{POWERS}(1+\epsilon)$, it is easy to see that H contains a $(1+\epsilon, 1)$ preserver of each detour terminating at v.

Now consider any index $i \in [2, k]$. Let us assume that the claim holds true for all indices $j < i$. Let v be a vertex in S_{k-i+1} and D be a detour terminating at v. We need to show that H contains a $(1+\epsilon, i)$ preserver of D. Let α be the smallest power of $(1+\epsilon)$ greater than or equal to $wt^*(D)$. Consider the detour $\text{HD}_\alpha(v)$. If $\sigma(\text{HD}_\alpha(v)) \cap S_{i+1} = \emptyset$, then H will contain the complete detour $\text{HD}_\alpha(v)$, which is a $(1+\epsilon, 1)$ preserver of D. And, we know that a $(1+\epsilon, 1)$-preserver of D is also a $(1+\epsilon, i)$-preserver of D. If $\sigma(\text{HD}_\alpha(v)) \cap S_{i+1} \neq \emptyset$, then H will contain the suffix $\text{HD}_\alpha(v)[z, v]$, where z is the last vertex in $\sigma(\text{HD}_\alpha(v))$ that lies in set S_{i+1}. Also, by induction hypothesis, H will

contain a $(1+\epsilon, i-1)$ preserver of prefix $\text{HD}_\alpha(v)[\cdot, z]$, say P. Now, by Definition 4.1, it follows that $P::\text{HD}_\alpha(v)[z, v]$ is a $(1+\epsilon, i)$ preserver of detour D, which is contained in H. This shows that the claim holds for index i as well. □

Lemma 4.4 along with Lemma 4.5 implies that upon failure of any vertex x, the shortest paths from s in graph $G\backslash\{x\}$ are stretched by a factor of at most $(1+\epsilon)^k$ in $H\backslash\{x\}$. We now do the analysis of the size of subgraph H.

Lemma 4.6. *With high probability,* $|H| = O(k \times n^{1+1/k} \log n \times \log_{1+\epsilon}(nW))$.

Proof. It is easy to see that for any $i \in [1, k]$, $|S_i| = O(\frac{n^{1+1/k}}{n^{i/k}})$. Consider the collection defined below.

$$\mathcal{C} = \{\sigma(\text{HD}_\alpha(v)) \mid v \in S_i, \alpha \in \text{POWERS}(1+\epsilon)\}$$

The collection $\mathcal{C}$ will contain at most n^2 sets. As for any vertex v, we can have at most n detours corresponding to n ancestors of v in T. From Lemma 2.3, it follows that for each $\sigma(\text{HD}_\alpha(v))$ of size greater than $n^{i/k} \log n$, w.h.p., the last $n^{i/k} \log n$ vertices of $\sigma(\text{HD}_\alpha(v))$ will contain a vertex from S_{i+1}. In other words, w.h.p., for each $v \in S_i$ and each $\alpha \in \text{POWERS}(1+\epsilon)$, $\text{SUFFIX}(\sigma(\text{HD}_\alpha(v)), S_{i+1})$ is of size at most $n^{i/k} \log n$. So, for any vertex v in S_i, we add at most $n^{i/k} \log n \times \log_{1+\epsilon}(nW)$ edges in Step 7. Since $|S_i| = O(\frac{n^{1+1/k}}{n^{i/k}})$, and i ranges from 1 to k, w.h.p., H contains $O(k \times n^{1+1/k} \log n \times \log_{1+\epsilon}(nW))$ edges. □

We thus get the following lemma.

Lemma 4.7. *Let G be a directed weighted graph on n vertices with maximum edge weight W and s being the designated source vertex. Then, we can compute in polynomial time a subgraph H satisfying the following: for any $x, v \in V$, $dist_{H\backslash x}(s, v) \leq (1+\epsilon)^k dist_{G\backslash x}(s, v)$. The size of H is $O(kn^{1+1/k} \log(n) \log_{1+\epsilon}(nW))$ with high probability.*

On substituting $\epsilon_{new} = \epsilon_{old}/(2k)$, and $k = \log_2(n)$ in Lemma 4.7, we get the following theorem.

Theorem 4.8. *Let G be a directed weighted graph on n vertices with maximum edge weight W and s being the designated source vertex. Then, we can compute in polynomial time a subgraph H satisfying the following relation: for any $x, v \in V$, $dist_{H\backslash x}(s, v) \leq (1+\epsilon) dist_{G\backslash x}(s, v)$. The size of H is $O(n \log^3(n) \log_{1+\epsilon}(nW))$ with high probability.*

5 PRECURSOR TO A COMPACT ROUTING

In the last section, we showed a construction of a sparse subgraph H that contained a $(1+\epsilon, k)$-preserver of each detour in G. In this section, we slightly modify the construction of subgraph H, which will help us later in obtaining an efficient routing scheme.

Recall that a $(1+\epsilon, k)$-preserver in H is a concatenation of at most k suffixes of detours, which were added to H in the k distinct rounds of Algorithm 4.2. Our routing scheme upon failure of any node x will route packets along these suffixes. In order to efficiently route along a suffix of a $(1+\epsilon, k)$ preserver, we require that each vertex on the suffix knows its successor on the suffix. Now if the frequency of a vertex w in all the suffixes included in H by Algorithm 4.2 is small, then the routing table required at w will be small. Thus, as a precursor to routing, we require that the frequency of any vertex in the suffixes added to H is bounded. More formally, if $Q_1, \ldots, Q_t$ are the suffixes added in a round i of Algorithm 4.2, then we need that the frequency of each vertex in $\{\sigma(Q_1), \ldots, \sigma(Q_t)\}$ is small.

Though Algorithm 4.2 ensures that the average frequency of a vertex is $\widetilde{O}(\log_{1+\epsilon}(nW))$, it fails to provide any bound on the maximum frequency. Recall that in any round, say i, of Algorithm 4.2, we compute the next subset S_{i+1} using random sampling. Instead of building this hierarchy of

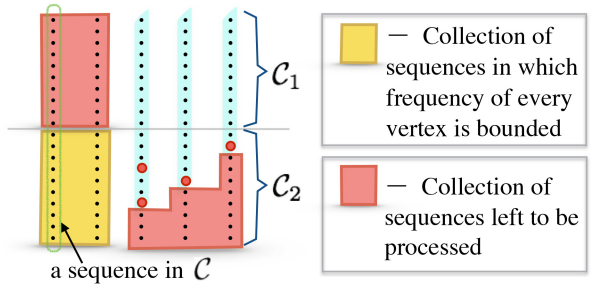

Fig. 3. Divide and conquer approach: vertices in red color represent the set S.

subsets randomly, we present in this section a deterministic algorithm that employs more insight into the structure of the suffixes.

We now explain our algorithm for deterministically computing the set S_{i+1} from the set S_i. The input to the algorithm is the collection $C = \{\sigma(\text{HD}_\alpha(v)) \mid v \in S_i, \alpha \in \text{POWERS}(1+\epsilon)\}$ and a parameter d to be fixed later on. Given a sequence σ, let FIRST-HALF(σ) and SECOND-HALF(σ), respectively, denote the subsequences of σ obtained by splitting it at midpoint. Our first step is to compute collections C_1 and C_2 by splitting each $\sigma \in C$ at midpoint, and adding FIRST-HALF(σ) to C_1 and SECOND-HALF(σ) to C_2. Next, we repeat the following two steps as long as there exists a vertex whose frequency in C_2 is greater than $d \log n$: (i) Pick such a vertex (say w) and add it to set S; (ii) Remove from C_2 each sequence σ in which w is present. It easy to see that set S will be of size at most $|C_2|/(d \log n) = |C|/(d \log n)$.

ALGORITHM 5.1: DET-CONSTRUCTION(C, d)

```
1  C_1 ← {FIRST-HALF(σ) | σ ∈ C};
2  C_2 ← {SECOND-HALF(σ) | σ ∈ C};
3  S ← ∅;
4  while ∃ w ∈ V s.t. FREQ(w, C_2) > d log n do
5  |   S ← S ∪ {w};
6  |   Remove all those σ from C_2 that contains vertex w;
7  C_new ← ∅;
8  foreach σ ∈ C do
9  |   if SECOND-HALF(σ) ∩ S ≠ ∅ then add SUFFIX(σ, S) to C_new;
10 |   if SECOND-HALF(σ) ∩ S = ∅ then add FIRST-HALF(σ) to C_new;
11 Return (S ⋃ DET-CONSTRUCTION(C_new, d));
```

Consider any sequence $\sigma \in C$. Let us first consider the case when SECOND-HALF(σ) $\cap S$ is non empty. (See Figure 3). If SUFFIX(σ, S) has no vertex of large frequency, then we are done with this sequence. However, it is quite possible that SUFFIX(σ, S) may still have vertices of large frequency. But, in this case, we are left to take care of only SUFFIX(σ, S), whose length is at most $|\sigma|/2$, for the computation of the set S_{i+1}. We now show how to take care of those sequences whose SECOND-HALF does not contain any vertex from S. Let C' denote the set of all those σ in C for which SECOND-HALF(σ) $\cap S$ is empty. Notice that the frequency of each vertex in the collection $\{\text{SECOND-HALF}(\sigma) \mid \sigma \in C'\}$ is bounded by $d \log n$. So, for any $\sigma \in C'$, we can safely discard SECOND-HALF(σ) and consider only FIRST-HALF(σ) for the computation of S_{i+1}. Thus, in this case, also, the length of sequence has been reduced to at most half.

The above discussion motivates us to design a recursive algorithm that performs at most $\log n$ recursions to obtain the desired set S_{i+1}. In Algorithm 5.1, we present the pseudocode of our construction. Since in each recursive call of DET-CONSTRUCTION, the set S included into S_{i+1} is of size at most $|C|/(d \log n)$, the size of the set S_{i+1} is at most $|C|/d$. Also, in each recursive call, the frequency of vertices increases by at most $d \log n$; thus, the set S_{i+1} will satisfy the condition that frequency of each vertex in $\{\text{SUFFIX}(\sigma, S_{i+1}) \mid \sigma \in C\}$ is bounded by $d \log^2 n$.

Recall that in Algorithm 4.2, $|S_i| \leq n/n^{(i-1)/k}$, and, thus, $|C| \leq n/n^{(i-1)/k} \log_{1+\epsilon}(nW)$. By our construction in Algorithm 5.1, we will have that $|S_{i+1}|$ is at most $|C|/d$, which we required to be bounded by $n/n^{i/k}$. This shows that d must be $n^{1/k} \log_{1+\epsilon}(nW)$. Finally, notice that the frequency of each vertex v in the set of all suffixes added during round i will be bounded by $d \log^2 n = n^{1/k} \log_{1+\epsilon}(nW) \log^2 n$. We thus have the following lemma.

LEMMA 5.1. *Let G be a directed graph on n vertices. There exists a construction for sets $S_1, S_2, \ldots, S_k$ in Algorithm 4.2 satisfying the following conditions.*

(1) For any index i, $|S_{i+1}| \leq n/n^{i/k}$.
(2) For any index i, frequency of each vertex in $\{\text{SUFFIX}(\sigma(HD_\alpha(v)), S_{i+1}) \mid v \in S_i, \alpha \in \text{POWERS}(1+\epsilon)\}$ is at most $n^{1/k} \log^2 n \log_{1+\epsilon}(nW)$.

5.1 A Sparse Subgraph with Bounded In-Degree

We here show that the alternative construction of sets $S_1, \ldots, S_k$ also gives a bound on the in-degree of each vertex in the sparse subgraph H. Notice that in Algorithm 4.2, an incoming edge is added to a vertex w only if $w \in \text{SUFFIX}(\sigma(\text{HD}_\alpha(v)), S_{i+1})$ for some v in S_i and $\alpha \in \text{POWERS}(1+\epsilon)$. So the number of incoming edges added to a vertex in round i is exactly equal to frequency of that vertex in $\{\text{SUFFIX}(\sigma(HD_\alpha(v)), S_{i+1}) \mid v \in S_i, \alpha \in \text{POWERS}(1+\epsilon)\}$. Therefore, Lemma 5.1 gives a bound of $kn^{1/k} \log^2 n \log_{1+\epsilon}(nW)$ on the in-degree of each vertex in the sparse subgraph, preserving distances $(1+\epsilon)^k$ approximately. On substituting $\epsilon_{new} = \epsilon_{old}/(2k)$ and $k = \log_2 n$, we get the following theorem.

THEOREM 5.2. *Let G be a directed weighted graph on n vertices with maximum edge weight W and s be the designated source vertex. Then, in polynomial time, we can compute a sparse subgraph H satisfying the following.*

(1) For any $x, v \in V$, $dist_{H \setminus x}(s, v)$ is less than or equal to $(1+\epsilon) dist_{G \setminus x}(s, v)$.
(2) The in-degree of each vertex in graph H is at most $O(\log^4(n) \log_{1+\epsilon}(nW))$.

6 AN ORACLE AND A LABELING SCHEME

We first describe a fault tolerant oracle for reporting approximate distances from s. Let v be the query vertex and x be the failed vertex. If x is not an ancestor of v, then the shortest path is the tree path $\text{PATH}_T(s, v)$, which remains intact in $G \setminus x$. So let us consider the more interesting case in which x is an ancestor of v. From Lemma 2.2, it follows that one of the shortest paths to v in $G \setminus x$ is of the form - $\text{PATH}_T(s, a) :: D :: \text{PATH}_T(b, v)$, where D is a detour avoiding x, and its weight is $dist_G(s, v) + wt^*(D)$.

Notice that if α_0 is the smallest power of $(1+\epsilon)$ for which there exists a vertex $b_0 \in \text{PATH}_T(\bar{x}, v)$ with $\text{HD}_{\alpha_0}(b_0)$ starting from an ancestor of x in T, then the value $dist_G(s, v) + \alpha_0$ would be a $(1+\epsilon)$ approximation of $dist_{G \setminus x}(s, v)$. Thus, in order to compute an approximation of $dist_{G \setminus x}(s, v)$, we need to compute this α_0 efficiently. We now describe our data structure that accomplishes this task.

For each $\alpha \in \text{POWERS}(1+\epsilon)$, we create a copy of T and denote it by T_α. For each vertex $y \in T_\alpha$, we set the weight of edge $(parent_{T_\alpha}(y), y)$ as $depth_T(\text{FIRST}_\alpha(y))$. Thus, the edge weights in any

tree T_α are integers in the range $[0, n-1]$. Notice that for any $\alpha \in \text{POWERS}(1+\epsilon)$, there will exist a detour starting from an ancestor of x and terminating at a vertex of $\text{PATH}_T(\bar{x}, v)$ with wt^* at most α if and only if the weight of the minimum weight edge on $\text{PATH}_{T_\alpha}(\bar{x}, v)$ is smaller than $depth_T(x)$. The smallest such α can be easily computed using the Bottleneck Edge Query (BEQ) data structure of Demaine et al. [13] for each of the trees T_α.

THEOREM 6.1 (DEMAINE ET AL. [13]). *Given a tree on n vertices with edge weights from the range $[0, n]$, it is possible to create in $O(n)$ time a data structure of $O(n)$ size so that given any $u, v \in V$, the edge of smallest weight on the unique path from u to v in the tree can be found in $O(1)$ time.*

In Algorithm 6.1, we present the pseudocode for determining approximate $s - v$ distance in $G \setminus x$. Since the BEQ query on a single tree can be answered in $O(1)$ time, the time for querying all the trees will be $O(\log_{1+\epsilon}(nW))$. However, instead of linearly checking all the powers of $(1+\epsilon)$ in the increasing order, if we perform a binary search on $\text{POWERS}(1+\epsilon)$, then query time is improved to $O(\log_2 \log_{1+\epsilon}(nW))$.

ALGORITHM 6.1: Determining $(1+\epsilon)$ approximate distance of v from s in graph $G \setminus \{x\}$.

```
1 if (x is not an ancestor of v), then
2     Return dist_G(s, v);
3 i ← depth_T(x);
4 for α ∈ POWERS(1 + ε) in increasing order do
5     j ← BEQ(x, v, T_α);
6     if j < i, then Return (dist_G(s, v) + α);
7 Return "Unreachable";
```

Finally, notice that the space complexity is $O(n \log_{1+\epsilon}(nW))$. The following theorem stems from the above discussion.

THEOREM 6.2. *Given a directed graph G on n vertices with maximum edge weight W and a source vertex $s \in V$, it is possible to compute a data structure of $O(n \log_{1+\epsilon}(nW))$ size that for any failing vertex x and any destination vertex v, there reports a $(1+\epsilon)$ approximation of the distance of v from s in $G \setminus \{x\}$ in $O(\log_2 \log_{1+\epsilon}(nW))$ time.*

6.1 Compact Labeling Scheme for Oracle

We now present the labeling scheme for oracle. Let v be a query vertex, and x be the failed vertex. Recall Algorithm 6.1. Our first step is to check whether x is an ancestor of v in T. One simple method to achieve this is to perform the pre-order and the post-order traversal of T, and store the pre-order as well as the post-order numbering of each vertex in its label. Now x will be an ancestor of v in T if and only if the pre-order numbering of x is smaller than that of v, and the post-order numbering of x is greater than that of v. Next, we assign i as $depth_T(x)$, extracting out this is also easy since depth information can also be made part of the label.

Thus, the only thing that is left is to obtain a labeling scheme for the BEQ problem. Demaine et al. [13] showed that the BEQ problem on any tree T_α is reducible to the LCA problem on a Cartesian tree, say $\mathcal{T}_\alpha$, which is related to tree T_α as follows.

(1) The vertices of T_α constitute the leaves of $\mathcal{T}_\alpha$.
(2) The edges of T_α constitute the internal nodes of $\mathcal{T}_\alpha$.
(3) For any two vertices $u, v \in T_\alpha$, the least weight edge on $\text{PATH}_{T_\alpha}(u, v)$ is the LCA of the leaf nodes corresponding to u and v in $\mathcal{T}_\alpha$.

Hence, the bottleneck edge query for any two vertices u and v in T_α can be answered by performing an LCA query for leaves u and v in $\mathcal{T}_\alpha$. However, notice that given the labels of u and v in tree $\mathcal{T}_\alpha$, we are not interested in computing the label of the edge stored at the LCA of u and v; rather, we are interested in knowing the weight of the edge stored at the LCA.

Alstrup et al. [1] established a labeling scheme for LCA that, given the labels of any two vertices u and v in a tree, returns a predefined label associated with the LCA of u and v in the tree. If each predefined label consist of M-bits, then the labels in this scheme for LCA consists of $O(M \log n_0)$ bits, where n_0 is the number of nodes in the tree. In our case, the number of nodes in $\mathcal{T}_\alpha$ is $O(n)$ only. Also, the predefined labels of internal nodes in $\mathcal{T}_\alpha$ stores just the depth value, thus M is $O(\log n)$.

Therefore, the label of any vertex v stores: (i) the pre-order and the post-order numbering of v, (ii) the depth of v in T, and (iii) the label of v corresponding to LCA queries in each tree $\mathcal{T}_\alpha$, where $\alpha \in \textsc{powers}(1+\epsilon)$. Hence, the label of each vertex consists of $O(\log^2 n \log_{1+\epsilon}(nW))$ bits. We thus have the following theorem.

Theorem 6.3. *Given a directed graph G on n vertices with maximum edge weight W and a source vertex $s \in V$, it is possible to compute vertex labels of $O(\log^2 n \log_{1+\epsilon}(nW))$ bits such that for any failing vertex x and any destination vertex v, the $(1+\epsilon)$ approximate distance of v from s in $G \setminus \{x\}$ can be determined by processing the labels associated with v and x only.*

Notice that in the above construction, the predefined label of an internal node of $\mathcal{T}_\alpha$, which corresponds to an edge in T_α, say $(parent_{T_\alpha}(y), y)$, just stores the value $depth_T(\textsc{first}_\alpha(y))$. However, we can also store in the predefined label other relevant information associated with either the detour $\text{HD}_\alpha(y)$, or a $(1+\epsilon, k)$ preserver of $\text{HD}_\alpha(y)$. Then, the result of Alstrup et al. [1] implies the following lemma, which would be used in the construction of the routing scheme.

Lemma 6.4. *For each y in V and $\alpha \in \textsc{powers}(1+\epsilon)$, let $\Phi_\alpha(y)$ be the M-bit information associated with a $(1+\epsilon, k)$ preserver of $HD_\alpha(y)$. Then, there exists a labeling scheme with labels of $O(M \log n \log_{1+\epsilon}(nW))$ bits such that given the label of any two vertices $x, v \in V$, where x is an ancestor of v in T, the following information can be retrieved.*

(i) Smallest $\alpha \in \textsc{powers}(1+\epsilon)$ such that there exists a vertex $b \in \textsc{path}_T(\bar{x}, v)$ whose $HD_\alpha(b)$ starts from an ancestor of x,

(ii) Vertex b stated in (i), and the M-bit information, i.e., $\Phi_\alpha(b)$, associated with the $(1+\epsilon, k)$ preserver of $HD_\alpha(b)$.

7 A COMPACT ROUTING SCHEME

Let v be a query vertex and x be a failed vertex. Let us consider the case in which x is an ancestor of v in T. Let D be the detour corresponding to the shortest s-v path in $G \setminus x$, and let α be the smallest power of $(1+\epsilon)$ greater than or equal to $wt^*(D)$. So there must exist a vertex $b \in \textsc{path}_T(\bar{x}, v)$ such that $\text{HD}_\alpha(b)$ starts from an ancestor of x. Such a vertex can be easily extracted using the labels of v and x described in Section 6. Since we can not afford to store the information of $\text{HD}_\alpha(t)$ explicitly for each $t \in V$, we make use of a $(1+\epsilon, k)$-preserver of $\text{HD}_\alpha(b)$ to route the packets to destination v.

Recall that a $(1+\epsilon, k)$-preserver of an $\text{HD}_\alpha(b)$ can be seen as a concatenation of at most $\ell (\leq k)$ paths, say $Q_\ell, Q_{\ell-1}, Q_2, Q_1$. (See Figure 4 (i)). If $b_{\ell+1}, b_\ell, \ldots, b_2, b_1$ are the endpoints of these paths, then

(i) For each $i < \ell$, Q_i is a suffix of detour to b_i that is added in the i^{th} round of Algorithm 4.2,

(ii) Q_ℓ is a detour to b_ℓ added in the ℓ^{th} round.

Also, notice that for each $i \in [1, \ell]$, vertex b_i belongs to set S_i. Now, for $i \in [1, \ell]$, let us denote by (c_i, d_i) the first non-tree edge in Q_i. (See Figure 4 (ii)). So, d_i is also the first vertex in $\sigma(Q_i)$.

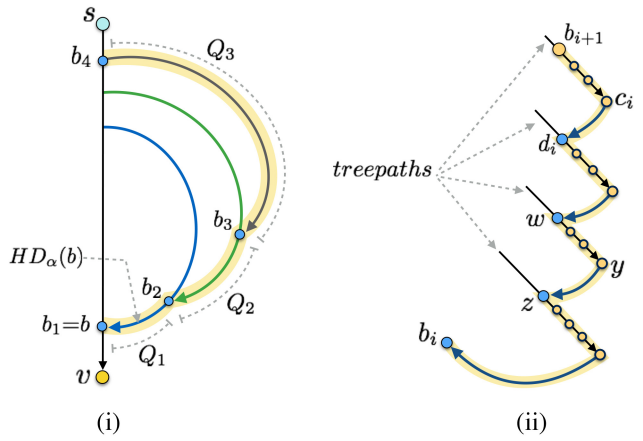

Fig. 4. (i) Depiction of segments Q_1, Q_2, Q_3 and vertices b_1, b_2, b_3, b_4 in a $(1+\epsilon, 3)$-preserver of $\text{HD}_\alpha(b)$. (ii) A finer description of Q_i showing the first non-tree edge (c_i, d_i).

We first extend the labeling scheme described in Section 6 to obtain a labeling scheme for routing.

7.1 Labeling Scheme for Routing

The label of each node consists of two subcomponents L_0 and L_1 as described below.

(1) In Ref. [25], Thorup and Zwick gave a construction of labeling scheme for rooted trees with $O(\log n)$ bit labels such that given the labels of any two nodes w and y (with w being an ancestor of y), one can compute the port number of the child of w on $\text{PATH}_T(w, y)$. The first component of the labels, i.e., L_0, would comprise of these labels. This will facilitate easy routing of packets along the tree paths. Also, the component L_0 would store the pre-order number, the post-order number, and the depth of vertex in T, so that ancestor descendant relationships can be easily verified. Thus, the label L_0 consists of $O(\log n)$ bits.

(2) The second component of the labels, that is L_1, will comprise of the labels described in Lemma 6.4, with $\Phi_\alpha(y)$ of a vertex y storing at most $k+1$ vertices and k non-tree edges associated with a $(1+\epsilon, k)$ preserver of $\text{HD}_\alpha(y)$, described above. (See Figure 4). So given the labels of v and x, we can extract out the value α equal to the smallest power of $(1+\epsilon)$ for which there exists a vertex $b \in \text{PATH}_T(\bar{x}, v)$ with $\text{HD}_\alpha(b)$ starting from an ancestor of x in T. Also, we can compute the L_0 label of vertices $b_{\ell+1}, b_\ell, \ldots, b_1$, and edges $(c_\ell, d_\ell), \ldots, (c_1, d_1)$ associated with the $(1+\epsilon, k)$-preserver of $\text{HD}_\alpha(b)$. Finally, notice that $\Phi_\alpha(y)$ consists of $O(k \log n)$ bits of information; thus, the label L_1 consists of $O(k \log^2(n) \log_{1+\epsilon}(nW))$ bits.

7.2 Description of Routing Table

We now give the construction of the routing table of a vertex $w \in V$. The routing table of w stores: (i) the label $L_0(w)$ and (ii) k segments, where the i^{th} segment corresponds to the i^{th} round of Algorithm 4.2. Below, we describe these k segments.

Let us fix an $i \in [1, k]$. Consider the detour $\text{HD}_\alpha(u)$ for some vertex $u \in S_i$ and an $\alpha \in \text{POWERS}(1+\epsilon)$. Let $u' \in S_{i+1}$ be the last vertex in $\sigma(\text{HD}_\alpha(u)) \cap S_{i+1}$, if it exists, else let u' be $\text{FIRST}_\alpha(u)$. Then, the suffix $\text{HD}_\alpha(u)[u', u]$ is added to H in the i^{th} round of Algorithm 4.2. Now, let us suppose $w \in \sigma\big(\text{HD}_\alpha(u)[u', u]\big)$, and let (y, z) be the first non-tree edge appearing after w

on $\text{HD}_\alpha(u)$. So z is the successor of w in the sequence $\sigma(\text{HD}_\alpha(u))$. Through our routing table, we need to ensure that if a packet reaches w, then it can easily find the route to vertex z. This route will be just the concatenation $\text{PATH}_T(w, y)::(y, z)$. So, corresponding to the triplet (i, u', u), in the routing table of w, we need to store the label of the edge (y, z).

Finally, notice that in Lemma 5.1 we showed that the frequency of each vertex in the set of all suffixes added in a round is $O(n^{1/k} \log^2(n) \log_{1+\epsilon}(nW))$. To store a non-tree edge (y, z), it suffices to store just the labels $L_0(y)$ and $L_0(z)$ that are of $O(\log n)$ bits. Thus, the size any of the k segments of the routing table of vertex w will be $O(n^{1/k} \log^3(n) \log_{1+\epsilon}(nW))$ bits, and the size of the routing table will be $O(kn^{1/k} \log^3(n) \log_{1+\epsilon}(nW))$ bits.

7.3 Routing Algorithm

We now describe the routing algorithm. Let $j \in [1, \ell]$ be the maximal index such that x is an ancestor of vertices $b_1, \ldots, b_j$. It follows from the proofs of Lemma 4.4 and Lemma 4.5 that the concatenation $P = \langle \text{PATH}_T(s, b_{j+1})::(Q_j, \ldots, Q_1)::\text{PATH}_T(b, v) \rangle$ is a path from s to v in $H \setminus x$ whose length is at most $(1+\epsilon)^k$ times the length of the shortest s-v path in $G \setminus x$. In our routing scheme, the packets from s to v will traverse this path P.

ALGORITHM 7.1: Route$(w, v, j, \langle b_\ell, \ldots, b_1, c_\ell, \ldots, c_1, d_\ell, \ldots, d_1 \rangle$, nextEdge$)$

```
1  if (w = v), then Return "Packet received";
2  if (j = 0 or w = b_1), then
3  |  w_new ← child of w on PATH_T(w, v);
4  |  Route(w_new, v, 0, <>,null);
5  if (nextEdge = null), then
6  |  if (w = b_j), then
7  |  |  nextEdge ← (c_{j-1}, d_{j-1});
8  |  |  j ← j − 1;
9  |  else
10 |  |  nextEdge ← Routing-Table-Look-Up(w, j, b_{j+1}, b_j);
11 (y, z) ← nextEdge;
12 if (w ≠ y), then
13 |  w_new ← child of w on PATH_T(w, y);
14 else
15 |  w_new = z;
16 |  nextEdge = null;
17 Route(w_new, v, j, ⟨b_ℓ, ..., b_1, c_ℓ, ..., c_1, d_ℓ, ..., d_1⟩, nextEdge);
```

Before starting the routing process, we use the labels of x and v to compute the vertices $b_1, \ldots, b_j$ and the edges $(c_1, d_1), \ldots, (c_j, d_j)$; also, this information is added to the header of the packet. Now, path P can be seen as a sequence of segments of tree paths joined through non-tree edges. So, whenever we reach any vertex $w \in \sigma(P)$, our first step is to calculate the next non-tree edge, say (y, z), appearing after w in P. Once this edge is computed, we traverse the tree path from w to y using L_0 labels. Once we reach y, the packet traverses the edge (y, z). On reaching any vertex $w \in \sigma(P)$, the next non-tree edge, say (y, z), on the path P can be computed as follows.

(1) If w is an internal vertex of some Q_i, $i \in [1, j]$, then we can just use the routing table stored at w to find the edge (y, z).

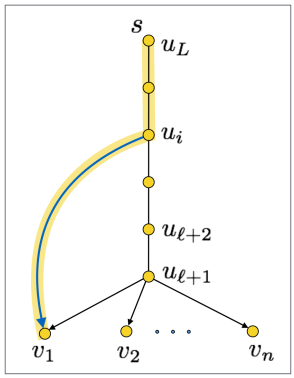

Fig. 5. The path highlighted in yellow color represents path $P_{i,1}$.

(2) If $w = b_i$ for some $1 < i \leq j$, the next non-tree edge will be (c_{i-1}, d_{i-1}) lying in segment Q_{i-1}. This information can be retrieved from the header of the packet itself.

We described above how to navigate from a vertex w in $\sigma(P)$ to the next vertex in $\sigma(P)$. Notice that (c_j, d_j) is the first non-tree edge in P. So to reach the first vertex d_j in $\sigma(P)$, path $\text{PATH}_T(s, c_j)::(c_j, d_j)$ can be traversed using L_0 labels. Finally, when we reach the last vertex in $\sigma(P)$, that is b_1, then, again, we use L_0 labels to traverse path $\text{PATH}_T(b_1, v)$ to reach vertex v. The pseudocode of this routing procedure is described in Algorithm 7.1.

Notice that the size of each label is $O(k \log^2(n) \log_{1+\epsilon}(nW))$ bits, and the size of each routing table is $O(kn^{1/k} \log^3(n) \log_{1+\epsilon}(nW))$ bits. Also, the size of the header attached to the packets is $O(k \log n)$ bits, and the stretch achieved is $(1+\epsilon)^k$. On substituting $\epsilon_{new} = \epsilon_{old}/(2k)$ and $k = \log_2(n)$, we get the following theorem.

Theorem 7.1. *For a directed network, there exists a fault tolerant scheme for routing packets from a fixed source vertex s with the following properties.*

(1) The label of each vertex consists of $O(\log^4(n) \log_{1+\epsilon}(nW))$ bits and the routing table consists of $O(\log^5(n) \log_{1+\epsilon}(nW))$ bits.
(2) While routing, each packet has to have extra $O(\log^2 n)$ bits as a header.
(3) To route packets from s to a destination v under failure of a vertex x, s should know labels (identity) of both v and x.
(4) The route taken by packets have a stretch of at most $(1+\epsilon)$ times that of the shortest path possible in $G \setminus x$.

8 LOWER BOUNDS

Let ϵ, W, n be such that ϵ lies in interval $(0, 1)$, and $W, n > 1$. We first show the construction of a graph G on $O(n)$ vertices with edge weights in range $[1, 2W]$ such that its $(1+\epsilon)$-distance preserving subgraph requires at least $\Omega(n \min\{n, \log_{1+\epsilon} W\})$ edges.

8.1 A Lower Bound on the Size of Subgraph

Let L, ℓ be integers to be fixed later on. We construct a graph G on $n + L - \ell$ vertices as follows. The vertex set of G is $\{u_{\ell+1}, u_{\ell+2}, \ldots, u_L, v_1, \ldots, v_n\}$, and the edge set of G is the union of the following two sets (see Figure 5).

(1) $E_1 = \{(u_L, u_{L-1}), \ldots, (u_i, u_{i-1}), \ldots, (u_{\ell+2}, u_{\ell+1})\}$,
(2) $E_2 = \{(u_i, v_j) \mid i \in [\ell+1, L], j \in [1, n]\}$.

We set $s := u_L$ as the designated source vertex. We now define our weight over the edges of graph G. For each $e \in E_1$, we set $wt(e) = 1$. For each $e \in E_2$, if u_i is the originating vertex of e, then we set $wt(e) = i + (1+2\epsilon)^i$. It is easy to see that the set $E_1 \cup \{(u_{\ell+1}, v_j) \mid j \in [1, n]\}$ constitutes the edges of the unique shortest path tree from s.

Now, for any $i \in [\ell+1, L]$ and $j \in [1, n]$, let $P_{i,j}$ denote the path $\text{PATH}_T(s, u_i)::(u_i, v_j)$ (see Figure 5). Then, for any j and any $i > \log_{1+2\epsilon}(L)$,

$$\frac{wt(P_{i+1,j})}{wt(P_{i,j})} = \frac{L + (1+2\epsilon)^{i+1}}{L + (1+2\epsilon)^i} > (1+\epsilon)$$

We set $L := \min\{n, \lfloor \log_{1+2\epsilon} W \rfloor\}$ and $\ell := \lfloor \log_{1+2\epsilon}(L) \rfloor$. Thus, G contains at most $2n$ vertices and all edge weights are in the range $[1, 2W]$.

Since i is always greater than $\ell = \lfloor \log_{1+2\epsilon}(L) \rfloor$, it follows that if vertex u_{i-1} fails, then the only $(1+\epsilon)$-approximate route to vertex v_j ($j \in [1, n]$) is path $P_{i,j}$. Hence, each v_j must keep all its incoming edges in the subgraph preserving approximate distances. There are $L - \ell = \Omega(L) = \Omega(\min\{n, \log_{1+\epsilon} W\})$ edges for each v_j. Thus, we are able to show that there exists a graph on $O(n)$ vertices with edge weights in range $[1, 2W]$ such that its $(1+\epsilon)$-distance preserving subgraph requires at least $\Omega(n \min\{n, \log_{1+\epsilon} W\})$ edges. So we have the following theorem.

THEOREM 8.1. *For any positive integers n, W and any $\epsilon > 0$, there exists an n-vertex directed graph with a source vertex s and edge weights in range $[1, W]$, whose 1-fault tolerant subgraph preserving distances from s up to a $(1+\epsilon)$ factor must have $\Omega(n \min\{n, \log_{1+\epsilon} W\})$ edges.*

8.2 A Lower Bound on the Size of Oracle

We now modify the construction of G (used in Section 8.1) to obtain a lower bound on the size of $(1+\epsilon)$-approximate distance reporting oracle. Let $Z_1, \ldots, Z_n$ be any arbitrary n vectors in $\{0,1\}^{L-\ell}$. We modify the weights of edges lying in the set E_2 as follows. For each $i \in [\ell+1, L]$ and each $j \in [1, n]$, we increase $wt(u_i, v_j)$ to value $(i + W)$, if the $(i-\ell)^{th}$ bit of vector Z_j is one.

Consider failure of a vertex u_{i-1} for some index i, $(\ell + 1 < i < L)$. The following two statements holds.

(1) If the $(i-\ell)^{th}$ bit of Z_j is one, then the length of the shortest path from s to v_j in $G \setminus u_{i-1}$ will be at least $L + (1+2\epsilon)^{i+1}$.
(2) If the $(i-\ell)^{th}$ bit of Z_j is zero, then path $P_{i,j}$ will be the unique shortest-path from s to v_j in $G \setminus u_{i-1}$. So, in this case, the $(1+\epsilon)$-approximate distance of v_j from s in $G \setminus u_{i-1}$ will be at most $(1+\epsilon) \times (L + (1+2\epsilon)^i)$, which is strictly less than $L + (1+2\epsilon)^{i+1}$.

This shows that by querying a $(1+\epsilon)$-approximate distance oracle we can extract out all $(L-\ell)$ bits of arbitrarily chosen vectors $Z_1, \ldots, Z_n$. Thus the oracle must contain at least $n(L-\ell)$ bits or $(n(L-\ell)/\log n)$ words. Hence, we get a lower bound of $\Omega(n \min\{n, \log_{1+\epsilon} W\}/\log n)$ on the size of the oracle.

We conclude with the following theorem.

THEOREM 8.2. *For any positive integers n, W and any $\epsilon > 0$, there exists an n-vertex directed graph with a source vertex s and edge weights in range $[1, W]$, whose 1-fault tolerant oracle reporting $(1+\epsilon)$ stretched distances from s must have $\Omega(n \min\{n, \log_{1+\epsilon} W\}/\log n)$ size.*

REFERENCES

[1] Stephen Alstrup, Esben Bistrup Halvorsen, and Kasper Green Larsen. 2014. Near-optimal labeling schemes for nearest common ancestors. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, January 5–7, 2014*. 972–982.

[2] Surender Baswana and Neelesh Khanna. 2013. Approximate shortest paths avoiding a failed vertex: Near optimal data structures for undirected unweighted graphs. *Algorithmica* 66, 1 (2013), 18–50.

[3] Aaron Bernstein and David Karger. 2008. Improved distance sensitivity oracles via random sampling. In *SODA'08: Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 34–43.

[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, May 31– June 2, 2009*. 101–110.

[5] Davide Bilò, Keerti Choudhary, Luciano Gualà, Stefano Leucci, Merav Parter, and Guido Proietti. 2018. Efficient oracles and routing schemes for replacement paths. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France (LIPIcs)*, Rolf Niedermeier and Brigitte Vallée (Eds.), Vol. 96. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 13:1–13:15.

[6] Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. 2016. Compact and fast sensitivity oracles for single-source distances. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22–24, 2016, Aarhus, Denmark (LIPIcs)*, Piotr Sankowski and Christos D. Zaroliagis (Eds.), Vol. 57. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 13:1–13:14.

[7] 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, February 17–20, 2016, Orléans, France*. 18:1–18:14.

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

[9] Greg Bodwin, Fabrizio Grandoni, Merav Parter, and Virginia Vassilevska Williams. 2017. Preserving distances in very faulty graphs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10–14, 2017, Warsaw, Poland*. 73:1–73:14.

[10] Shiri Chechik. 2013. Fault-tolerant compact routing schemes for general graphs. *Inf. Comput.* 222 (2013), 36–44.

[11] Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. 2010. Fault tolerant spanners for general graphs. *SIAM J. Comput.* 39, 7 (2010), 3403–3423.

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

[13] Erik D. Demaine, Gad M. Landau, and Oren Weimann. 2014. On Cartesian trees and range minimum queries. *Algorithmica* 68, 3 (2014), 610–625.

[14] 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.

[15] Michael Dinitz and Robert Krauthgamer. 2011. Fault-tolerant spanners: Better and simpler. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, June 6–8, 2011*. 169–178.

[16] Ran Duan. 2011. *Algorithms and Dynamic Data Structures for Basic Graph Optimization Problems*. Ph.D. Dissertation. University of Michigan.

[17] Ran Duan and Seth Pettie. 2009. Dual-failure distance and connectivity oracles. In *SODA'09: Proceedings of 19th Annual ACM -SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 506–515.

[18] Manoj Gupta and Shahbaz Khan. 2017. Multiple source dual fault tolerant BFS trees. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10–14, 2017, Warsaw, Poland*. 127:1–127:15.

[19] Manoj Gupta and Aditi Singh. 2018. Generic single edge fault tolerant exact distance oracle. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9–13, 2018, Prague, Czech Republic*. 72:1–72:15.

[20] J. Moy. 1999. *OSPF: Anatomy of an Internet Routing Protocol*. Addison-Wesley.

[21] Enrico Nardelli, Guido Proietti, and Peter Widmayer. 2003. Swapping a failing edge of a single source shortest paths tree is good and fast. *Algorithmica* 35, 1 (2003), 56–74.

[22] Merav Parter. 2015. Dual failure resilient BFS structure. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21–23, 2015*. 481–490.

[23] Merav Parter and David Peleg. 2013. Sparse fault-tolerant BFS trees. In *Algorithms - ESA 2013-21st Annual European Symposium, Sophia Antipolis, France, September 2–4, 2013. Proceedings*. 779–790.

[24] David Peleg and Alejandro A. Schäffer. 1989. Graph spanners. *J. Graph Theory* 13, 1 (1989), 99–116.
[25] Mikkel Thorup and Uri Zwick. 2001. Compact routing schemes. In *SPAA*. 1–10.
[26] Jan van den Brand and Thatchaphol Saranurak. 2019. Sensitive distance and reachability oracles for large batch updates. *CoRR* abs/1907.07982 (2019).

Received November 2019; revised April 2020; accepted April 2020