Downloaded 09/03/20 to 94.223.125.90. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

$(1+\epsilon)$-Approximate f-Sensitive Distance Oracles

Shiri Chechik [*] Sarel Cohen [†] Amos Fiat [‡] Haim Kaplan [§]

Abstract

An f-Sensitive Distance Oracle with stretch α preprocesses a graph $G(V, E)$ and produces a small data structure that is used to answer subsequent queries. A query is a triple consisting of a set $F \subset E$ of at most f edges, and vertices s and t. The oracle answers a query (F, s, t) by returning a value $\tilde{d}$ which is equal to the length of some path between s and t in the graph $G \setminus F$ (the graph obtained from G by discarding all edges in F). Moreover, $\tilde{d}$ is at most α times the length of the shortest path between s and t in $G \setminus F$. The oracle can also construct a path between s and t in $G \setminus F$ of length $\tilde{d}$. To the best of our knowledge we give the first nontrivial f-sensitive distance oracle with fast query time and small stretch capable of handling multiple edge failures. Specifically, for any $f = o(\frac{\log n}{\log \log n})$ and a fixed $\epsilon > 0$ our oracle answers queries (F, s, t) in time $\widetilde{O}(1)$ with $(1+\epsilon)$ stretch using a data structure of size $n^{2+o(1)}$. For comparison, the naïve alternative requires $m^f n^2$ space for sublinear query time.

1 Introduction.

Dealing with failures is an essential part of modern computing. Built in processes that deal with failures are an essential part of many computing environments, from massive storage devices, large scale parallel computation, and communication networks.

In this paper we study how to answer queries in the presence of failures in a network, where one assumes apriori a bound on the number of simultaneous failures. Such a bound is especially relevant in the context of independent failures where the probability of multiple failures decreases exponentially with the number of failures. In particular, we consider distance and shortest path queries in the presence of edge failures for undirected weighted graphs.

Our goal is to preprocess a graph $G = (V, E)$ and produce some (hopefully small) data structure used to answer subsequent queries of the form (F, s, t), $F \subset E$, $|F| \leq f$. The answer to such a query is an approximation to the length of the shortest path between s and t in the graph $G \setminus F$ (the graph obtained from G by discarding all edges in F). Moreover, we want to be able to answer queries that ask to compute such an approximate shortest path. We call such a data structure an f-sensitive distance oracle.

There are several parameters to be considered with respect to suggested f-sensitive distance oracles.

- The size of the data structure.
- The time needed to answer distance queries (F, s, t).
- The approximation ratio (also known as the stretch), how close are our estimated distances to the actual distances?
- The time required for the preprocessing phase.

The f-sensitive distance oracle is said to have a stretch of α if the estimated distance returned by the data structure is at most α times the actual distance. *I.e.*, for every set $F \subset E$ such that $|F| \leq f$ and for every pair of vertices s and t, the f-sensitive distance oracle returns an estimated distance $\widehat{\mathrm{d}}_{G\setminus F}(s,t)$, that satisfies $\mathrm{d}_{G\setminus F}(s,t) \leq \widehat{\mathrm{d}}_{G\setminus F}(s,t) \leq \alpha \mathrm{d}_{G\setminus F}(s,t)$ (where $\mathrm{d}_{G\setminus F}(s,t)$ is the distance from s to t in the graph $G \setminus F$).

The naïve solution to the exact problem is to store for every vertex v and for every set F with $|F| \leq f$, a shortest path tree rooted by v and a distance vector giving the distances from v to all other vertices. Thus, the total space required by the naïve solution is $O(m^f n^2)$. Using the naïve approach, a query (F, s, t) returns the exact solution and requires $\widetilde{O}(f)$ time (to

[*]Tel Aviv University, Israel. E-mail: shiri.chechik@gmail.com. This research was supported by the Israel Science Foundation grant No. 1528/15, the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11) and the Blavatnik Fund.

[†]Tel Aviv University, Israel. E-mail: sarelcoh@post.tau.ac.il. This research was supported by the Israel Science Foundation grant No. 1528/15, Israel Science Foundation grant No. 1841/14, the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11), the Blavatnik Fund and GIF grant No. 1161.

[‡]Tel Aviv University, Israel. E-mail: fiat@tau.ac.il. This research was supported by Israel Science Foundation grant No. 1841/14, the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11) and the Blavatnik Fund.

[§]Tel Aviv University, Israel. E-mail: haimk@post.tau.ac.il. This research was supported by Israel Science Foundation grant No. 1841/14, the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11), the Blavatnik Fund and GIF grant No. 1161.

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 09/03/20 to 94.223.125.90. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

find the appropriate distance vector). To print out the path one also needs to follow the predecessor links in the appropriate shortest path tree. This can be done in time proportional to the number of edges of the shortest path.

Another naïve solution requires linear space and also returns the exact solution. This solution does not preprocess the graph, and upon query (F, s, t) run Dijkstra's algorithm from s in the graph $G \setminus F$. This is space efficient, but query time is high.

In section 2 we present a simple exact solution with query time $O(f^2)$ and space $O(n^2 \cdot n^f)$. The primary difficulty that we address is how to replace the n^f factor in this space bound, by roughly $\log^f n \cdot 1/\epsilon^f$. We achieve this at the cost of returning an approximate shortest path rather than an exact shortest path (with stretch $1+\epsilon$), and slightly increasing the query time. The basic idea that we use to reduce the size of the data structure is to consider sets of edges (called segments) rather than singleton edges. More specifically, instead of considering the deletion of each edge individually, we consider the deletion of the entire segment containing the edge. When done appropriately, the deletion of segments rather than individual edges, maintains approximate shortest paths.

Related Work: The problem of f-sensitive distance oracles is closely related to dynamic shortest paths algorithms. In dynamic algorithm as opposed to the f-sensitive model, the failures/updates are not given as a batch but rather as an adversarial sequence of update and query operations. The problem of dynamic shortest path was extensively studied since the early 80's (e.g. [13, 20, 21, 22, 3, 9, 24, 25, 23, 5, 4, 16, 15, 18, 17, 1]) and for many problems we have dynamic algorithms with bounds close to optimal. The situation with the f-sensitive model is perhaps surprisingly very different and many key problems in this model are far from being fully understood.

The first distance sensitive oracle was in the context of directed graphs [10]. It maintained exact distances and was capable of handling a single edge failure. The space requirement of this oracle is $O(n^2 \log n)$ and its query time is $O(\log n)$.

This was later generalized to handle a single vertex or edge failure in [11]. Demetrescu et al. [11] presented an exact 1-sensitive distance oracle of size $O(n^2 \log n)$, $O(1)$ query time and $\widetilde{O}(mn^2)$ preprocessing time.

Later, in two consecutive papers, Karger and Bernstein improved the preprocessing time (while keeping the space and query time unchanged), first in to $O(n^2\sqrt{m})$ in [6] and then to $\widetilde{O}(mn)$ in [7].

Afterwards, by a quite involved construction, Duan and Pettie [12] considered the case of two failures (vertices or edges) with exact distances. The size of their oracle is $O(n^2 \log^3 n)$, the query time is $O(\log n)$ and the construction time is polynomial. They claim: "It may yet be possible to find distance oracles capable of handling any fixed number of failures. However, the sheer complexity of our algorithm suggests that moving beyond dual-failures will require a fundamentally different approach to the problem". Further details on their dual-failure data-structure appear in Duan's PhD Thesis [27].

Khanna and Baswana [19] considered approximate 1-sensitive distance oracles for unweighted graphs. More precisely, they presented a data structure of size $O(k^5 n^{1+1/k} \frac{\log^3 n}{\epsilon^4})$, $(2k-1)(1+\epsilon)$ stretch and $O(k)$ query time.

The problem of 1-sensitive distance oracles was also studied with a special focus on improving the preprocessing time at the cost of a polynomial query time (see [26, 14]).

The case of handling more than two edge failures (while keeping query time small) was also considered in [8] — albeit the stretch was $\Omega(f)$ which is quite large. Let W denote the weight of the heaviest edge in the graph assuming the smallest edge weight is 1. For any integer parameter k, [8] gave an f-sensitive distance oracle of size $O(fkn^{1+1/k} \log(nW))$, $(8k-2)(f+1)$ stretch and $O(f \cdot \log^2 n \cdot \log\log^2 n)$ query time. Note that even with $k = 1$ and for a constant number of failures, the stretch is quite large, and therefore irrelevant in many settings.

Our Results: To the best of our knowledge, we present the first non trivial f-sensitive distance oracle with small stretch and fast query time capable of handling multiple edge failures for weighted undirected graphs. We are unaware of any previous construction that deals with more than 2 failures and gives stretch arbitrarily close to one (See Tables 1, 2). We deal with an arbitrary (predetermined) upper bound on the number of failures, and arbitrarily small (constant) stretch. More precisely, our main result is the following theorem.

Theorem 1.1. *Let $G = (V, E)$ be a weighted undirected graph, with $|V| = n$, and $|E| = m$, where all edges have positive weight in the range $[1, W]$. Let $D = n \cdot W$ (which is an upper bound on the diameter of the graph), and let ϵ be a parameter such that $0 < \epsilon < 1$. One can construct in $O\left(\left(fn^5 + n^4 \log D + n^3 \log D \log\log D\right)(\log D/\epsilon)^f\right)$ time an f-sensitive distance oracle for G with $(1+\epsilon)$ stretch, $O\left(n^2 (\log D/\epsilon)^f \cdot f \log D\right)$ space, and $O\left(f^5 \log D\right)$ query time.*

In addition, we show that one can replace the $\log^f D$

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Model	f - Max # failures	Approximation	Data Structure Size	Query time	Preprocessing Time	Ref.
Directed Edge Failure	1	1	$O(n^2 \log n)$	$O(\log n)$	$O(mn^2 \log n + n^3 \log^2 n)$	[10]
Directed Edge/vertex Failure	1	1	$O(n^2 \log n)$	$O(1)$	$O(mn^2)$	[11]
Directed Edge/vertex Failure	1	1	$O(n^2 \log n)$	$O(1)$	$\widetilde{O}(mn)$	[6], [7]
Directed Edge/vertex Failure	2	1	$O(n^2 \log^3 n)$	$O(\log n)$	poly(n)	[12]
Undirected Unweighted Edge/vertex Failure	1	$(2k-1)(1+\epsilon)$ $k \geq 1$, $\epsilon > 0$	$O\left(k^5 n^{1+1/k} \log^3 n/\epsilon^4\right)$	$O(k)$	$O\left(kmn^{1+1/k}\right)$	[19]
Undirected Weighted Edge Failure	$f \in Z$	$(8k-2)(f+1)$ $k \geq 1$	$O\left(fkn^{1+1/k} \log(nW)\right)$	$O\left(f \cdot \log^2 n \cdot \log\log^2 n\right)$	poly(n)	[8]

Table 1: Previous f-sensitive Distance Oracles. Note that if the number of failures $f > 2$ then the stretch is $(8k-2)(f+1) \geq 24$.

Model	f - Max # failures	Approximation	Data Structure Size	Query time	Preprocessing Time	Ref.
Undirected Unweighted Edge Failure	$f \in Z$	$1+\epsilon$ $\epsilon > 0$	$O\left(n^3 (\log n/\epsilon)^f\right)$	$O(f^4)$	$O(n^2(\log n/\epsilon)^f(m + n\log n))$	This Paper
Undirected Unweighted Edge Failure	$f \in Z$	$1+\epsilon$ $\epsilon > 0$	$O\left(n^2 (\log n/\epsilon)^f \cdot f \log n\right)$	$O\left(f^5 \log n\right)$	$O\left(\left(fn^5\right)(\log n/\epsilon)^f\right)$	This Paper
Undirected Weighted Edge Failure	$f \in Z$	$1+\epsilon$ $\epsilon > 0$	$O\left(n^3 (\log D/\epsilon)^f\right)$	$O(f^4)$	$O(n^2(\log D/\epsilon)^f(m + n\log n))$	This Paper
Undirected Weighted Edge Failure	$f \in Z$	$1+\epsilon$ $\epsilon > 0$	$O\left(n^2 (\log D/\epsilon)^f \cdot f \log D\right)$	$O\left(f^5 \log D\right)$	$O\left(\left(fn^5 + n^4 \log D + n^3 \log D \log\log D\right)(\log D/\epsilon)^f\right)$	This Paper
Undirected Weighted Edge Failure	$f \in Z$	$1+\epsilon$ $\epsilon > 0$	$O\left(n^3 (\log n/\epsilon)^f (\log W/\log n)\right)$	$O\left(f^4 \log\log W\right)$	$O\left(n^2 (\log n/\epsilon)^f (\log W/\log n)(m + n\log n)\right)$	This Paper
Undirected Weighted Edge Failure	$f \in Z$	$1+\epsilon$ $\epsilon > 0$	$O\left(n^2 (\log n/\epsilon)^f \cdot f \log W\right)$	$O\left(f^5 \log n \log\log W\right)$	$O\left(fn^5 (\log n/\epsilon)^f (\log W/\log n)\right)$	This Paper

Table 2: New f-sensitive Distance Oracles. Note that the stretch is now $1+\epsilon$ for arbitrary small $\epsilon > 0$. The first two rows describe our results for unweighted graphs, rows 3 and 4 give our results for weighted graphs as a function of $D = n \cdot W$, and the last two rows improve the space bound as a function of the max edge weight W.

factor in Theorem 1.1 by (roughly) $\log^f n$, at the cost of an additional $\log\log W$ factor in the query time.

The rest of the paper is organized as follows. In Section 2 we present an f-sensitive $(1+\epsilon)$ stretch distance oracle that requires $O(n^3(\log D/\epsilon)^f)$ space, and $O(f^4)$ query time, where $D = n \cdot W$ is an upper bound on the diameter of the graph (the edge weights are normalized so that the minimum edge has weight 1 and W is the maximum edge weight). In Section 3 we reduce the space requirements of our data structure to $O(n^2(\log D/\epsilon)^f \cdot f \log D)$ and obtain Theorem 1.1. The main idea used in Section 3 is that we do not need to explicitly store the paths in our data-structures. Rather we can represent the paths implicitly as the concatenation of at most $O(f \log D)$ shortest paths in the original graph G. We do this by developing new tools introduced in Section 3 (decomposable paths and expaths).

We further improve the space in Section 4 and replace the $\log^f D$ factor in Theorem 1.1 by (roughly) $\log^f n$ at the cost of an additional $\log\log W$ factor in the query time (see Table 2 for the full details).

1.1 Preliminaries.

We use the following notations:

Let P be a path. $|P|$ is defined to be the sum of the weights of its edges.

$\mathrm{d}_G(u,v)$ is the distance from u to v in G, abbreviated $\mathrm{d}(u,v)$ when G is known from the context. $P_G(u,v)$ denotes an arbitrary shortest path from u to v in G and is abbreviated $P(u,v)$ when G is known from the con-

Downloaded 09/03/20 to 94.223.125.90. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 09/03/20 to 94.223.125.90. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

text.

Let P be a path and x, y be two vertices along the path. Then $P[x, y]$ is the subpath of P from x to y (including x and y).

We now define the path concatenation $(\cdot)$ operator. Let $P_1 = (x_1, x_2, \ldots, x_r)$ and $P_2 = (y_1, y_2, \ldots, y_t)$ be two paths. Then $P = P_1 \cdot P_2$ is defined as the path $P = (x_1, x_2, \ldots, x_r, y_1, y_2, \ldots, y_t)$, and it is well defined if either $x_r = y_1$ or $(x_r, y_1) \in E$.

2 $\widetilde{O}(n^3)$ Approximate Distance Sensitive Oracles.

In this section we present an f-sensitive distance oracle for a given graph G with $(1+\epsilon)$ stretch, $O\left(n^3 (\log D/\epsilon)^f\right)$ space, and $O\left(f^4\right)$ query time.

The intuition for our construction comes from a simple data structure that can accommodate up to f edge failures and returns the exact shortest path in the graph without the failed edges. Unfortunately, the space requirements are quite large, $O(n^{f+3})$. We describe this simple data structure in Appendix A as the paper is self-contained without it, and we give it to provide some more intuition.

2.1 Improved Data Structure.

Similarly to the $O(n^{f+3})$ data structure, our $O(n^3(\log D/\epsilon)^f)$ data structure consists of rooted trees $FT(u, v)$ for every pair of vertices $u, v \in V$. We further improve upon the space requirement in Section 3.

To define $FT(u, v)$ we first define a general decomposition of a path P into subpaths using a particular subset of the vertices of P which we call netpoints. We use $\epsilon' = \epsilon/3$ in these definitions.

DEFINITION 2.1. (PATH NETPOINTS) *Let $P = (v_1 = u, v_2, \ldots, v_k = v)$ be a path from u to v in G. Let L be the set of all vertices $v_j, v_{j+1} \in P$ such that $|P[u, v_j]| < (1+\epsilon')^i \leq |P[u, v_{j+1}]|$ for some integer $i \geq 0$. Define R analogously to be the set of all vertices $v_j, v_{j-1} \in P$ such that $|P[v_j, v]| < (1+\epsilon')^i \leq |P[v_{j-1}, v]|$ for some integer i. Let* $\mathrm{netpoints}(P) = L \cup R \cup \{u, v\}$. *See Figure 1.*

Note that the cardinality of $\mathrm{netpoints}(P)$ is $O(\log_{1+\epsilon'} |P|) = O(\log_{1+\epsilon} |P|)$ and for a small enough ϵ, $O(\log_{1+\epsilon} |P|) = O(\frac{\log |P|}{\epsilon}) = O(\log D/\epsilon)$.

A *segment* of P is a subpath of P connecting two consecutive netpoints along P. We denote by $\mathrm{seg}(P)$ the set of segments of the path P. From this definition it follows that the segments in $\mathrm{seg}(P)$ are pairwise disjoint except possibly for their endpoints. For $e \in P$ we define $\mathrm{seg}(e, P)$ to be the segment of P containing e, see Figure 2.

The following lemma follows immediately from the definition of netpoints and segments.

LEMMA 2.1. *Let P be a path from u to v, $e = (x, y)$ be an edge of P. Either* $\mathrm{seg}(e, P) = \{e\}$ *or* $|\mathrm{seg}(e, P)| \leq \epsilon' \min\{|P[u, x]|, |P[x, v]|\}$.

Proof. Assume with out loss of generality that the vertex x is closer than y to u along P (*i.e.* $|P[u, x]| < |P[u, y]|$). Let i be the maximal index such that $(1+\epsilon')^i \leq |P[u, x]|$. Then $(1+\epsilon')^{i+1} > |P[u, x]|$. If $(1+\epsilon')^{i+1} \leq |P[u, y]|$ then we get $|P[u, x]| < (1+\epsilon')^{i+1} \leq |P[u, y]|$ and by definition both x and y are netpoints of P, hence $\mathrm{seg}(e, P) = \{e\}$. Otherwise, $(1+\epsilon')^{i+1} > |P[u, y]|$ and

$$|\mathrm{seg}(e, P)| \leq (1+\epsilon')^{i+1} - (1+\epsilon')^i = \epsilon'(1+\epsilon')^i \leq \epsilon' |P[u, x]|.$$

Symmetrically, we can show that either $\mathrm{seg}(e, P) = \{e\}$ or $|\mathrm{seg}(e, P)| \leq \epsilon' |P[y, v]|$. If $|\mathrm{seg}(e, P)| \leq \epsilon' |P[y, v]|$, it follows that $|\mathrm{seg}(e, P)| \leq \epsilon' |P[y, v]| < \epsilon' |P[x, v]|$ (since $|P[y, v]| < |P[x, v]|$ by assumption). Therefore, either $\mathrm{seg}(e, P) = \{e\}$ or $|\mathrm{seg}(e, P)| \leq \epsilon' \min\{|P[u, x]|, |P[x, v]|\}$.

The trees $FT(u, v)$. We associated with each node $\alpha \in FT(u, v)$ a particular graph G_α, the shortest path $P_\alpha := P_{G_\alpha}(u, v)$ which is the shortest path from u to v in the graph G_α, and a segment $S_\alpha \in \mathrm{seg}(P_{p(\alpha)})$ where $p(\alpha)$ is the parent of α in $FT(u, v)$. The graphs G_α are defined top down as follows.

The graph G_r associated with the root r of $FT(u, v)$ is the input graph $G = (V, E)$ itself (we formally define $S_r = \emptyset$). We introduce a child α of r for each segment $s \in \mathrm{seg}(P_r)$, define $S_\alpha = s$, and G_α to be the graph obtained from G_r by deleting the edges of s.

In general, assume that we have defined $FT(u, v)$ up to level k. We define level $k+1$ as follows. For each node α' of level k for which $P_{\alpha'} \neq \emptyset$ we introduce a child α, per segment $s \in \mathrm{seg}(P_{\alpha'})$, define $S_\alpha = s$, and G_α to be the graph obtained from $G_{\alpha'}$ by deleting the edges of s. Nodes α' of level k for which $P_{\alpha'} = \emptyset$ are leaves of $FT(u, v)$.

For each node $\alpha \in FT(u, v)$ we define $\mathrm{avoid}(\alpha)$ to be the set containing all edges in the segment associated with α and in the segments associated with the ancestors of α in $FT(u, v)$ (equivalently one can define recursively $\mathrm{avoid}(\alpha) = \mathrm{avoid}(p(\alpha)) \cup S_\alpha$). Then $G_\alpha = G \setminus \mathrm{avoid}(\alpha)$. See Figure 3.

Our data structure consists of the tree $FT(u, v)$ for every pair of vertices u, v, constructed up to level f. In each node $\alpha \in FT(u, v)$ we keep only the path P_α. We store the edges of P_α in a perfect hash table, keeping the index of $\mathrm{seg}(e, P_\alpha)$ with each edge e. This index identifies the child β of α such that $S_\beta = \mathrm{seg}(e, P_\alpha)$.

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 09/03/20 to 94.223.125.90. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Path netpoints and segments.

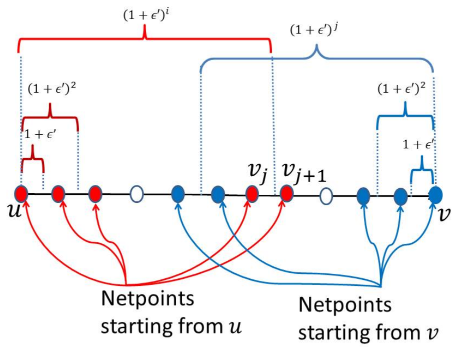


Figure 1: Path netpoints, the union of both sets of netpoints, from u (L in text) and from v (R in text). Although not shown, the same vertex can be chosen both for L and for R, and, moreover, the same vertex can be chosen multiple times for L and/or for R.

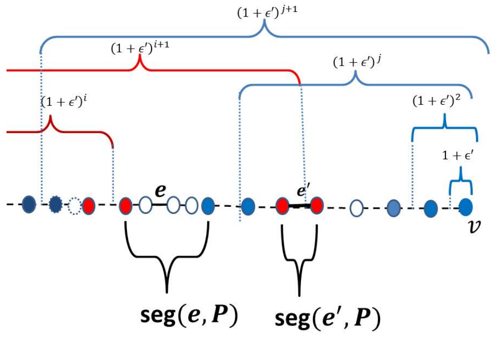


Figure 2: Segments of a path and seg(e, P).

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 09/03/20 to 94.223.125.90. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

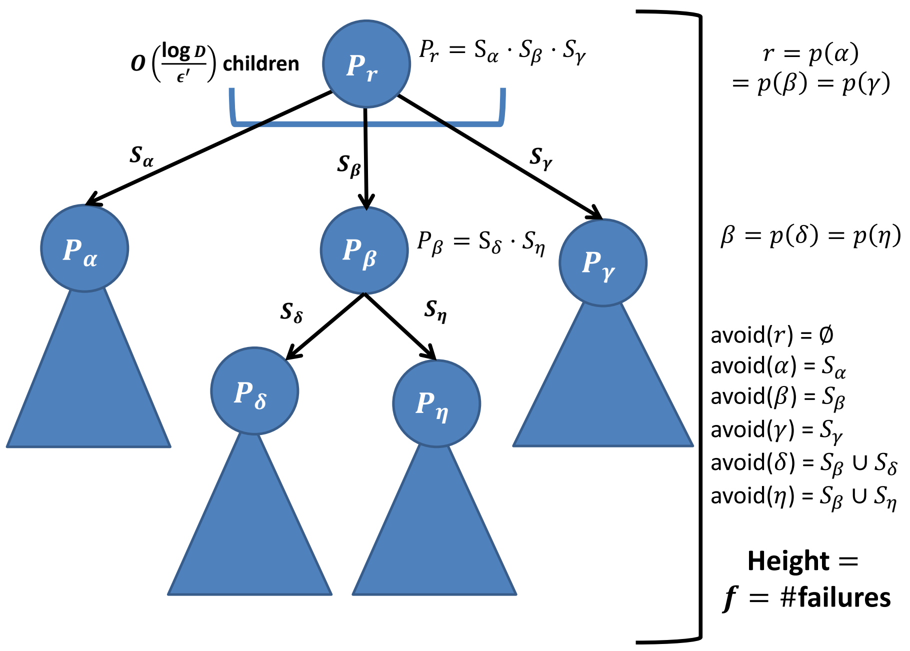

$O\left(\frac{\log D}{\epsilon'}\right)$ children
P_r
$P_r = S_\alpha \cdot S_\beta \cdot S_\gamma$
S_α
S_β
S_γ
P_α
P_β
$P_\beta = S_\delta \cdot S_\eta$
P_γ
S_δ
S_η
P_δ
P_η
$r = p(\alpha) = p(\beta) = p(\gamma)$
$\beta = p(\delta) = p(\eta)$
avoid(r) = $\emptyset$
avoid(α) = S_α
avoid(β) = S_β
avoid(γ) = S_γ
avoid(δ) = $S_\beta \cup S_\delta$
avoid(η) = $S_\beta \cup S_\eta$
Height = f = #failures

Figure 3: A tree $FT(u, v)$. The path at the root r is $P_r = P_G(u, v)$, the shortest path in the graph G between vertices u and v. Every node in $FT(u, v)$ has $O(\log D/\epsilon')$ children. The path P_r is split by the vertices of netpoints(P_r) into segments S_α, S_β, and S_γ. Associated with each such segment is a child of node r, nodes α, β, and γ. The path P_α is the shortest path in the graph G_α, where G_α is the result of deleting all edges along S_α from the graph $G = G_r$ (*i.e.*, deleting avoid(α)). Paths P_β and P_γ are defined analogously. The path P_β is split by netpoints(P_β) into segments P_δ, P_η. Segments P_δ and P_η have associated nodes δ and η — children of node β. The path P_δ is the shortest path between vertices u and v in the graph G_δ, where G_δ is the result of deleting the edges along S_β and S_δ from G. Equivalently, G_δ is the result of deleting the edges along S_δ from G_β. The path P_η is defined similarly.

Since the number of children of each node of $FT(u, v)$ is $O(\log D/\epsilon)$ and the depth of $FT(u, v)$ is f then $FT(u, v)$ contains $O((\log D/\epsilon)^f)$ nodes. Each node α keeps the path P_α in a data structure of size linear in number of edges of P_α which is $O(n)$. It follows that the total size of $FT(u, v)$ is $O(n(\log D/\epsilon)^f)$. The total size of the trees $FT(u, v)$ for all pairs $u, v \in V$ is $O(n^3(\log D/\epsilon)^f)$. This is the size of the data structure.

Preprocessing. We construct $FT(u, v)$ top down while maintaining G_α. At each node α we have to run a single source shortest path computation to find P_α. Subsequently we compute netpoints(P_α) and seg(P_α) in $O(n)$ time and construct the hash table storing the edges of P_α in $O(n)$ expected time. Summing up over all nodes α in all trees $FT(u, v)$ we get that the total preprocessing time is $O(n^2(\log D/\epsilon)^f(m+n\log n))$ expected time.

2.2 Query.

Given $F \subset E, |F| \leq f$ (a set of failing edges), and a pair of vertices $s, t \in V$, we approximate the length of the shortest path from s to t in $G \setminus F$ as follows.

The graph H: We build a small weighted graph H over $V(H) = V(F) \cup \{s, t\}$, *i.e.* the vertices of H are the endpoints of the edges of F, and s, t.[1] For each pair of vertices $u, v \in V(H)$ we decide if $E(H)$ contains (u, v) and assign it a weight $w_H(u, v)$ as follows.

We traverse the tree $FT(u, v)$ from its root towards a leaf. When visiting a node $\alpha \in FT(u, v)$, we do one of the following steps.

Case 1: If $P_\alpha = \emptyset$ we stop the traversal and don't add an edge between u, v in H.

Case 2: If $P_\alpha \cap F = \emptyset$ we add an edge (u, v) to H with $w_H(u, v) = |P_\alpha|$ and stop the traversal of $FT(u, v)$.

[1] We use the notation $V(F)$ for a subset of edges F, to denote the endpoints of the edges of F. We use the notation $V(H)$ for a graph H to denote the set of vertices of the graph.

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 09/03/20 to 94.223.125.90. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Case 3: Otherwise, P_α must contain a failing edge $e \in F$. We choose arbitrarily an edge $e \in P_\alpha \cap F$ and continue the traversal in a subtree of α obtained by following the child α' of α such that $S_{\alpha'} = \text{seg}(e, P_\alpha)$.

We prove in Lemma 2.3 below that this traversal is well defined in the sense that we never reach Case 3 when α is at level f. We add an edge to H only if this procedure ends in Case 2. So each edge (u, v) in H corresponds to some path in $G \setminus F$ and $w_H(u, v)$ is the length of this path.

Recall that $\text{avoid}(\alpha)$ is the set of edges along the segments $S_{\alpha'}$ for all nodes α' that are ancestors of α in $FT(u, v)$ (including edges of S_α), G_α is the graph $G \setminus \text{avoid}(\alpha)$ and P_α is the shortest path between u to v which avoids the edges $\text{avoid}(\alpha)$.

LEMMA 2.2. *During the traversal of $FT(u, v)$ let α_d be the node visited at depth d, it must be that $|\text{avoid}(\alpha_d) \cap F| \geq d$.*

Proof. By induction on d. For $d = 0$ this holds vacuously. For $d > 1$, we traversed the edge (α_{d-1}, α_d) labeled by $\text{seg}(e, P_{\alpha_{d-1}})$ where $e \in F \cap P_{\alpha_{d-1}}$. By definition,

$$\text{avoid}(\alpha_d) = \text{avoid}(\alpha_{d-1}) \cup \text{seg}(e, P_{\alpha_{d-1}}).$$

Since $e \in F \cap (\text{avoid}(\alpha_d) \setminus \text{avoid}(\alpha_{d-1}))$, we get that

$$|\text{avoid}(\alpha_d) \cap F| \geq |\text{avoid}(\alpha_{d-1}) \cap F| + 1,$$

and the lemma follows by induction.

LEMMA 2.3. *The query procedure is well-defined in the sense that it never applies Case 3 in a leaf of $FT(u, v)$.*

Proof. Let $\alpha \in FT(u, v)$ be a node that is traversed by the query in which we apply Case 3. In Case 3 we have $P_\alpha \cap F \neq \emptyset$, let $e \in P_\alpha \cap F$. Since $e \in P_\alpha$ it follows that $e \notin \text{avoid}(\alpha)$, thus $|\text{avoid}(\alpha) \cap F| < |F| = f$. By Lemma 2.2, it must be that the depth of α is less than f, and therefore α is not a leaf of $FT(u, v)$.

We use H to answer the query as follows. If s and t are disconnected in H, then we answer that s and t are also disconnected in $G \setminus F$. Otherwise, we compute and return $\text{d}_H(s, t)$ as our approximation of $\text{d}_{G \setminus F}(s, t)$.

LEMMA 2.4. *Query time is $O(f^4)$.*

Proof. This is the time it takes to build the graph H. For each pair $u, v \in V(H)$ (there are $O(f^2)$ such pairs) we traverse the tree $FT(u, v)$ along a root-to-leaf path (the length of the root-to-leaf path is at most f) and at each node along the root-to-leaf path we find a segment containing a failing edge. We show below how to find a segment containing a failing edge in each node which we traverse in $O(f)$ time. Therefore, the total time to build the graph H is $O(f^4)$ (that is, $O(f^2)$ trees $FT(u, v)$, multiplied by $O(f)$ nodes on a root-to-leaf path in each tree, multiplied by $O(f)$ time per node to find the appropriate segment). Finding the shortest path from s to t in H takes $\widetilde{O}(f^2)$ time, so the query time is dominated by the time required to build H.

To determine if $e \in P$ and to find $\text{seg}(e, P)$ if $e \in P$, we store the edges of P in a perfect hash table, keeping also the index of $\text{seg}(e, P)$ with each edge. This allows to determine if an edge e is in P in $O(1)$ time. To determine if there is an edge $e \in F$ that belongs to P we query the hash table with the edges of F one by one in $O(f)$ total time.

2.3 Correctness.

We prove that $\text{d}_H(s, t)$ is a good approximation of $\text{d}_{G \setminus F}(s, t)$.

The very high level idea of our analysis is that we distinguish between two cases. The first case is when all failures are "far away" from the shortest path from s to t in $G \setminus F$. In this case, we show that we can find an exact shortest path from s to t by querying the tree $FT(u, v)$ (see Lemma 2.5). Loosely speaking, since all failures are far away from the shortest path, discarding at each level of the search in $FT(u, v)$ a segment containing a failed edge doesn't eliminate any of the edges of P. Hence the shortest path survives during the query of $FT(u, v)$. The second case is when there is a "close by" failure to the shortest path from s to t in $G \setminus F$. In this case, we show that we can construct an approximate shortest path that consists only of sub-paths where such sub-path is a path between vertices in $V(F) \cup \{s, t\}$ and each sub-path is "far away" from all failures and thus can be found by querying the appropriate tree $FT(u, v)$.

To prove the correctness we use the following definition of the trapezoid of a path.

DEFINITION 2.2. ($\text{tr}_G(P)$) *Let P be an arbitrary path from u to v in G. Let $x \in V$ be a vertex and $\ell \geq 0$ a real number, define*

$$\text{ball}_G(x, \ell) = \{u \mid \text{d}_G(x, u) \leq \ell\}.$$

$\text{ball}_G(x, \ell)$ is the set of all vertices whose distance in G from x is at most ℓ. Define the trapezoid of P in G to be $\text{tr}_G(P) = \left(\bigcup_{x \in P \setminus \{u,v\}} \text{ball}_G(x, \epsilon' \cdot \min(|P[u, x]|, |P[x, v]|))\right) \setminus \{u, v\}$.

The next lemma gives a sufficient condition for having an edge (u, v) in H.

LEMMA 2.5. *Let $u, v \in V(H)$, and let P be an arbitrary*

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 09/03/20 to 94.223.125.90. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

path from u to v in $G \setminus F$. If

$$\mathrm{tr}_{G\setminus F}(P) \cap V(H) = \emptyset$$

then (u, v) is an edge of H and $w_H(u, v) \le |P|$.

Proof. Let $\alpha_0(= r), \alpha_1, \ldots, \alpha_k$, $k \le f$, be the nodes of $FT(u, v)$ that we traversed during a query. We prove below that P is contained in G_{α_i} for every $1 \le i \le k$. From this follows that the query concluded the traversal of $FT(u, v)$ in Case 2 (recall the cases of the procedure that decides if to add (u, v) to H in Section 2.2), added (u, v) to H, and set $w_H(u, v) = |P_{\alpha_k}| = \mathrm{d}_{G_{\alpha_k}}(u, v)$. Since P is a path between u and v in G_{α_k}, and P_{α_k} is the shortest path between u and v in G_{α_k}, we obtain that $w_H(u, v) \le |P|$.

The crux of the proof is to show that P is contained in G_{α_i} for $i = 0, \ldots, k$. We prove this by induction on i. The base case is obvious since $G_r = G$. To simplify the notation which we use to establish the induction step, let $\alpha = \alpha_i$ and $\alpha' = \alpha_{i+1}$. We assume that P is contained in G_α and we need to prove that P is contained in $G_{\alpha'}$. The only way P can disappear from $G_{\alpha'}$ is if there is a segment of P_α that contains an edge $e_F \in F$ and an edge $e_P \in P$. (Note that $e_F \ne e_P$ since $P \in G \setminus F$.)

Assume by contradiction that such a segment $s \in \mathrm{seg}(P_\alpha)$ exists and assume without loss of generality that there isn't any other edge of F or P between e_F and e_P in s, and thus $P_\alpha[x, y]$ is a path in $G \setminus F$. Let x be the endpoint of e_P which is closer to e_F along P_α, and let y be the endpoint of e_F which is closer to e_P along P_α. Note that $x, y \notin \{u, v\}$. See Figure 4.

The vertex x appears along P and partitions P into two subpaths $P[u, x]$ and $P[x, v]$. Similarly, the vertex x also appears along P_α and splits it into two subpaths $P_\alpha[u, x]$ and $P_\alpha[x, v]$. Since P_α is the shortest path from u to v in G_α and P is a path in G_α it follows that $|P_\alpha[u, x]| \le |P[u, x]|$ and $|P_\alpha[x, v]| \le |P[x, v]|$ which implies that

$$\min\{|P_\alpha[u, x]|, |P_\alpha[x, v]|\} \le \min\{|P[u, x]|, |P[x, v]|\}\ . \tag{2.1}$$

Since $x \in V(P)$, $y \in V(H)$, and $\mathrm{tr}_{G\setminus F}(P) \cap V(H) = \emptyset$ by the assumption of the lemma, it follows that $y \notin \mathrm{tr}_{G\setminus F}(P)$ and so (see Figure 5)

$$\mathrm{d}_{G\setminus F}(x, y) > \epsilon' \min\{|P[u, x]|, |P[x, v]|\}. \tag{2.2}$$

Combining Equations (2.1) and (2.2) and since $|P_\alpha[x, y]| \ge \mathrm{d}_{G\setminus F}(x, y)$ (since we assumed $P_\alpha[x, y]$ is a path in $G \setminus F$) we get that

$$|\mathrm{seg}(e_F, P_\alpha)| \ge |P_\alpha[x, y]| \ge \mathrm{d}_{G\setminus F}(x, y) >$$
$$\epsilon' \min\{|P[u, x]|, |P[x, v]|\} \ge \epsilon' \min\{|P_\alpha[u, x]|, |P_\alpha[x, v]|\}$$

Since $\mathrm{seg}(e_F, P_\alpha)$ contains both e_F, and e_P and $e_F \ne e_P$, it cannot be that $\mathrm{seg}(e_F, P_\alpha) = \{e_F\}$. Therefore we obtain a contradiction to Lemma 2.1, since neither $\mathrm{seg}(e_F, P_\alpha) = \{e_F\}$ nor $|\mathrm{seg}(e_F, P_\alpha)| \le \epsilon' \min\{|P_\alpha[u, x]|, |P_\alpha[x, v]|\}$.

When the condition of Lemma 2.5 does not hold (*i.e.*, $\mathrm{tr}_{G\setminus F}(P) \cap V(H) \ne \emptyset$) then the following lemma shows how to find a vertex z in $\mathrm{tr}_{G\setminus F}(P) \cap V(H)$ such that one of the distances from u to z or from z to v can be approximated using Lemma 2.5, and the other path is shorter than P, and hence can be approximated recursively. See Figure 7 where x is u, there is an edge from u to z in H according to Lemma 2.5, and the approximate length of the dashed path from z to v is found recursively. The proof of Theorem 2.1 below specifies this process, Lemmata 2.5 above and 2.6 below are used in the proof of Theorem 2.1.

LEMMA 2.6. *Let $u, v \in V(H)$, and let $P = P_{G\setminus F}(u, v)$. If $\mathrm{tr}_{G\setminus F}(P) \cap V(H) \ne \emptyset$ then there exist vertices x,y, and z such that (see Figure 6):*

1. *$x \in \{u, v\}, y \in P, z \in \mathrm{tr}_{G\setminus F}(P) \cap V(H)$.*

2. *$|P[x, y]| \le \frac{1}{2} |P[u, v]|$.*

3. *$\mathrm{d}_{G\setminus F}(y, z) \le \epsilon' \mathrm{d}_{G\setminus F}(x, y)$.*

4. *Let $P' = P_{G\setminus F}(x, y) \cdot P_{G\setminus F}(y, z)$. Then $\mathrm{tr}_{G\setminus F}(P') \cap V(H) = \emptyset$.*

Proof. Since $\mathrm{tr}_{G\setminus F}(P) \cap V(H) \ne \emptyset$, there exists $z \in \mathrm{tr}_{G\setminus F}(P) \cap V(H)$. By the definition of $\mathrm{tr}_{G\setminus F}(P)$ there exists $y \in P$ such that $z \in \mathrm{ball}_{G\setminus F}(y, \epsilon' \min\{|P[u, y]|, |P[y, v]|\})$. We then choose $x \in \{u, v\}$ such that $|P[x, y]| = \min\{|P[u, y]|, |P[y, v]|\}$ and obtain that x, y, z satisfy Conditions 1, 2 and 3.

Now among all triples of vertices x, y, z that satisfy Conditions 1, 2, and 3 we choose the triple x, y, z that minimizes $\mathrm{d}_{G\setminus F}(x, y)$, breaking ties in favor of the triple that minimizes $\mathrm{d}_{G\setminus F}(y, z)$. We show that these x, y, z satisfy Condition 4.

Assume by contradiction that $\mathrm{tr}_{G\setminus F}(P') \cap V(H) \ne \emptyset$. Then there are vertices $z' \in \mathrm{tr}_{G\setminus F}(P') \cap V(H)$ and $y' \in P'$ such that $\mathrm{d}_{G\setminus F}(y', z') \le \epsilon' \min\{|P'[x, y']|, |P'[y', z]|\}$. As $P' = P_{G\setminus F}(x, y) \cdot P_{G\setminus F}(y, z)$ and $y' \in P'$ then either $y' \in P_{G\setminus F}(x, y)$ or $y' \in P_{G\setminus F}(y, z)$.

If $y' \in P_{G\setminus F}(x, y) \setminus \{y\}$ then $|P[x, y']| < |P[x, y]|$ and this contradicts the choice of x, y, z. (We should have preferred x, y', z', see the upper part of Figure 6)

If $y' \in P_{G\setminus F}(y, z)$ (including the case $y' = y$), then

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 09/03/20 to 94.223.125.90. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

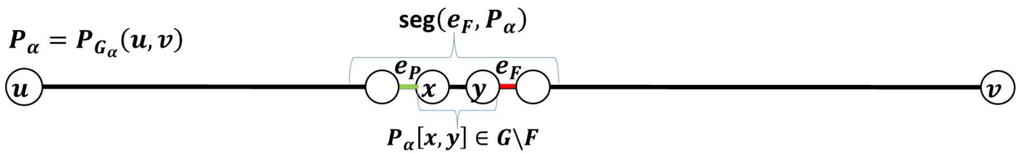

$P_\alpha = P_{G_\alpha}(u,v)$
$seg(e_F, P_\alpha)$
e_P
e_F
u
x
y
v
$P_\alpha[x,y] \in G\backslash F$

Figure 4: The path P_α and the edges e_P and e_F in the proof of Lemma 2.5.

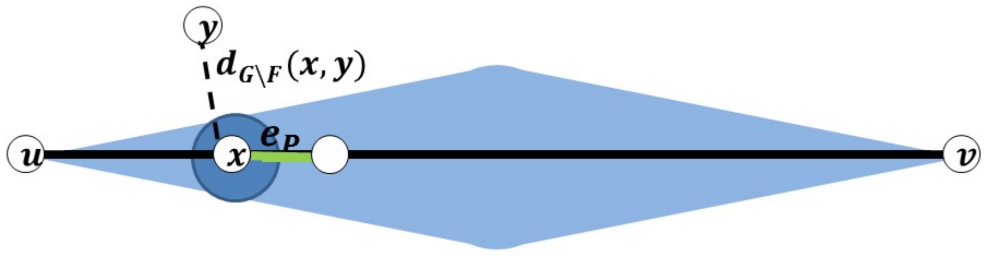


Figure 5: Proof of Lemma 2.5: Since $y \notin \mathrm{tr}_{G\setminus F}(P)$ we have $\mathrm{d}_{G\setminus F}(x,y) > \epsilon' \min\{|P[u,x]|, |P[x,v]|\}$.

$\mathrm{d}_{G\setminus F}(y',z') \le \epsilon' \mathrm{d}_{G\setminus F}(y',z)$ and therefore

$$\begin{aligned}
&\mathrm{d}_{G\setminus F}(y,z') \le \\
&\mathrm{d}_{G\setminus F}(y,y') + \mathrm{d}_{G\setminus F}(y',z') = \\
&(\mathrm{d}_{G\setminus F}(y,z) - \mathrm{d}_{G\setminus F}(z,y')) + \mathrm{d}_{G\setminus F}(y',z') \le \\
&\mathrm{d}_{G\setminus F}(y,z) - \mathrm{d}_{G\setminus F}(z,y') + \epsilon' \mathrm{d}_{G\setminus F}(y',z) < \\
&\mathrm{d}_{G\setminus F}(y,z) \ . \qquad (\text{assuming} \epsilon' < 1)
\end{aligned}$$

This also contradicts the choice of x, y, z. (We should have preferred x, y, z' since $\mathrm{d}_{G\setminus F}(y,z') < \mathrm{d}_{G\setminus F}(y,z)$. See the lower part of Figure 6).

We are now ready to establish the correctness of our query algorithm.

THEOREM 2.1. **(Correctness)** *Our algorithm determines if s and t are connected in $G \setminus F$ and if they are connected we have that (see Figure 7) $\mathrm{d}_{G\setminus F}(s,t) \le \mathrm{d}_H(s,t) \le (1+\epsilon)\mathrm{d}_{G\setminus F}(s,t)$.*

Proof. An edge (u,v) in H corresponds to a path P_α from u to v in $G \setminus F$ and $w_H(u,v) = |P_\alpha|$ (the path must be in $G \setminus F$ because the only possibility for an edge (u,v) to be added to H is due to case 2 in Section 2.2 of the query procedure, when building the graph H). Therefore, if there is a path from s to t in H of length ℓ', then there is also a path from s to t in $G \setminus F$ of length ℓ'. Thus, $\mathrm{d}_H(s,t) \ge \mathrm{d}_{G\setminus F}(s,t)$ and in particular if s and t are disconnected in $G \setminus F$ then they are also disconnected in H. So in the rest of the proof we assume that there is (at least one) path from s to t in $G \setminus F$.

We prove by induction on $\mathrm{d}_{G\setminus F}(u,v)$ that for every pair of vertices $u, v \in V(H)$ we have that $\mathrm{d}_{G\setminus F}(u,v) \ge \frac{\mathrm{d}_H(u,v)}{1+\epsilon}$ (or, equivalently, $\mathrm{d}_H(u,v) \le (1+\epsilon)\mathrm{d}_{G\setminus F}(u,v)$). We assume that for every $u', v' \in V(H)$ such that $\mathrm{d}_{G\setminus F}(u',v') < \mathrm{d}_{G\setminus F}(u,v)$ we have already established that $\mathrm{d}_{G\setminus F}(u',v') \ge \frac{\mathrm{d}_H(u',v')}{1+\epsilon}$, and prove the claim for u, v.

Let $P = P_{G\setminus F}(u,v)$ and consider the following two cases.

Case 1 $\mathrm{tr}_{G\setminus F}(P) \cap V(H) = \emptyset$**:** In this case it follows from Lemma 2.5 that $(u,v) \in E(H)$ and $w_H(u,v) \le |P| = \mathrm{d}_{G\setminus F}(u,v)$. So, $w_H(u,v) = \mathrm{d}_{G\setminus F}(u,v)$ and the lemma follows.

Case 2 $\mathrm{tr}_{G\setminus F}(P) \cap V(H) \ne \emptyset$**:** (See Figure 7) In this case by Lemma 2.6 there exist vertices x, y, z such that $x \in \{u, v\}$, $y \in V(P)$, and $z \in V(H) \cap \mathrm{tr}_{G\setminus F}(P)$, $\mathrm{d}_{G\setminus F}(y,z) \le \epsilon' |P[x,y]| = \epsilon' \mathrm{d}_{G\setminus F}(x,y)$, and $\mathrm{tr}(P_{G\setminus F}(x,y) \cdot P_{G\setminus F}(y,z)) \cap V(H) = \emptyset$. We assume for the rest of the proof that $x = u$. If $x = v$ then the proof is symmetric.

Since $\mathrm{tr}_{G\setminus F}(P(u,y) \cdot P_{G\setminus F}(y,z)) \cap V(H) = \emptyset$ it follows from Lemma 2.5 that $(u,z) \in E(H)$,

$$(2.3) \quad |w_H(u,z)| \le |P_{G\setminus F}(u,y) \cdot P_{G\setminus F}(y,z)| = dist_{G\setminus F}(u,y) + \mathrm{d}_{G\setminus F}(y,z) \ .$$

Using the triangle inequality in H and Equation

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 09/03/20 to 94.223.125.90. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

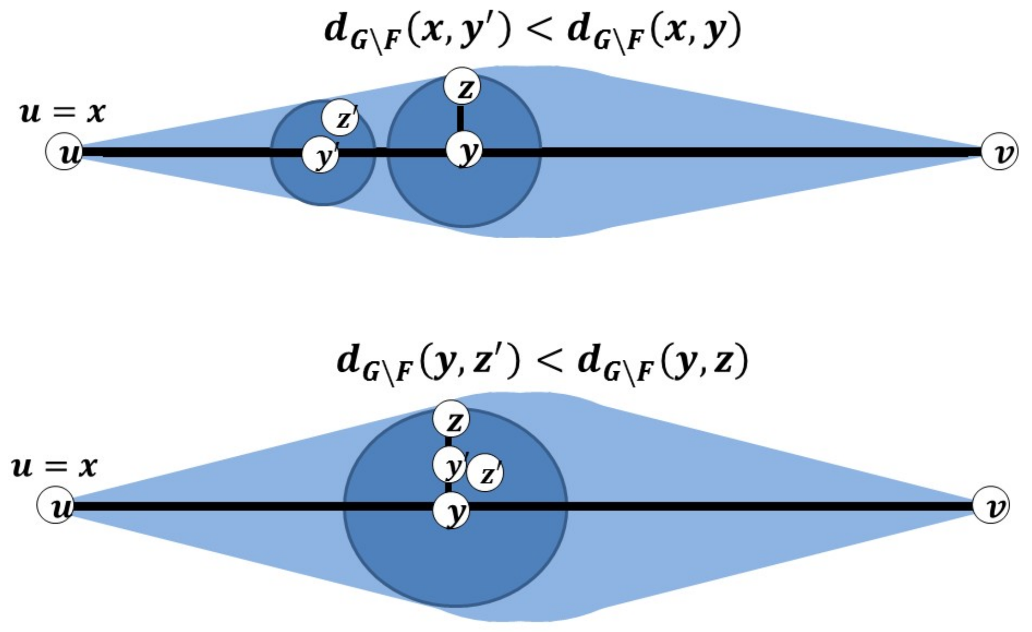

$d_{G\setminus F}(x,y') < d_{G\setminus F}(x,y)$
$u=x$
u
z'
y'
z
y
v
$d_{G\setminus F}(y,z') < d_{G\setminus F}(y,z)$
$u=x$
u
z
y'
z'
y
v

Figure 6: An illustration of the proof of Lemma 2.6.

(2.3) we have (see the edge (u,z) in Figure 7)

$$\text{(2.4)} \quad \mathrm{d}_H(u,v) \le w_H(u,z) + \mathrm{d}_H(z,v) \le \mathrm{d}_{G\setminus F}(u,y) + \mathrm{d}_{G\setminus F}(y,z) + \mathrm{d}_H(z,v)$$

Now we apply the induction hypothesis on $u' = z$ and $v' = v$. We can do so since every point in $\mathrm{tr}_{G\setminus F}(P) \cap V(H)$ is closer to v than to u and in particular, $\mathrm{d}_{G\setminus F}(z,v) < \mathrm{d}_{G\setminus F}(u,v)$ (see dashed edge in Figure 7). This application of the induction hypothesis together with Equation (2.4) gives

$$\text{(2.5)} \quad \mathrm{d}_H(u,v) \le \mathrm{d}_{G\setminus F}(u,y) + \mathrm{d}_{G\setminus F}(y,z) + (1+\epsilon)\mathrm{d}_{G\setminus F}(z,v)$$

Finally we apply the triangle inequality in $G\setminus F$ and the fact that $\mathrm{d}_{G\setminus F}(y,z) \le \epsilon' \mathrm{d}_{G\setminus F}(u,y)$ to obtain that

$$\begin{aligned}
&\mathrm{d}_H(u,v) \le \\
&\mathrm{d}_{G\setminus F}(u,y) + \epsilon' \mathrm{d}_{G\setminus F}(u,y) + (1+\epsilon)(\mathrm{d}_{G\setminus F}(z,y) + \mathrm{d}_{G\setminus F}(y,v)) \le \\
&\mathrm{d}_{G\setminus F}(u,y) + \epsilon' \mathrm{d}_{G\setminus F}(u,y) + (1+\epsilon)(\epsilon' \mathrm{d}_{G\setminus F}(u,y) + \mathrm{d}_{G\setminus F}(y,v)) \le \\
&\mathrm{d}_{G\setminus F}(u,y) + \epsilon'(2+\epsilon)\mathrm{d}_{G\setminus F}(u,y) + (1+\epsilon)\mathrm{d}_{G\setminus F}(y,v) \le \\
&(1+\epsilon)(\mathrm{d}_{G\setminus F}(u,y) + \mathrm{d}_{G\setminus F}(y,v)) \le \\
&(1+\epsilon)\mathrm{d}_{G\setminus F}(u,v)
\end{aligned}$$

where the next to last inequality holds since $\epsilon' < \epsilon/3 \le \epsilon/(\epsilon+2)$, and the last inequality holds since P is a shortest path between u and v in $G \setminus F$. Therefore, $\mathrm{d}_{G\setminus F}(u,v) \ge \frac{\mathrm{d}_H(u,v)}{1+\epsilon}$, and we proved the induction theorem.

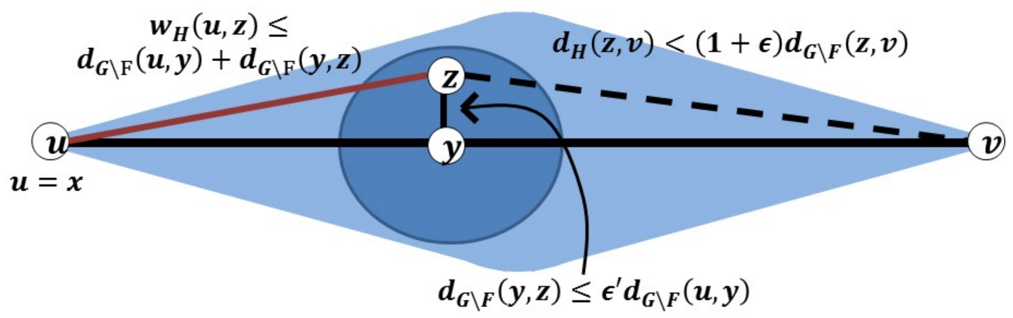
3 Reducing Space by Using Decomposable Paths.

Our data-structure in Section 2 requires $O(n^3(\log D/\epsilon)^f)$ space. In this section we develop a variant of this data structure which requires only $O(n^2(\log D/\epsilon)^f \cdot f \log D)$ space and keeps the query and preprocessing algorithms efficient. The main idea is a method to store the paths P_α implicitly rather than explicitly (in a hash table). To specify our new paths P_α and their representation we need the following definition.

Definition 3.1. (k-decomposable path) *Let G be a graph and $G \setminus A$ a subgraph of G obtained by discarding the edges in A. A k-decomposable path in $G \setminus A$ is a path which is the concatenation of at most $k+1$ shortest paths of G interleaved with at most k edges. Denote the shortest k-decomposable path from u to v in $G \setminus A$ by $P^k_{G\setminus A}(u,v)$.*

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 09/03/20 to 94.223.125.90. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php



$w_H(u,z) \leq$
$d_{G\setminus F}(u,y) + d_{G\setminus F}(y,z)$
$d_H(z,v) < (1+\epsilon)d_{G\setminus F}(z,v)$
z
u
y
v
$u = x$
$d_{G\setminus F}(y,z) \leq \epsilon' d_{G\setminus F}(u,y)$

Figure 7: An illustration of the proof of Theorem 2.1: (u, z) is an edge in H; we apply the induction hypothesis on z and v.

We also use the following Theorem of Afek et al. [2].

THEOREM 3.1. *Let G be an undirected graph and let F be a set of f edges of G. Each shortest path in $G \setminus F$ is an f-decomposable path.*

If we keep a representation of all shortest paths in G (which takes $O(n^2)$ space) then we can represent an f-decomposable path P using only $2f+2$ vertices which are the endpoints of the $f+1$ shortest paths and the f edges, whose concatenation is P. In particular, a shortest path in $G \setminus F$, for F of size at most f, can be represented using $2f+2$ vertices.

However when we try to apply this approach in our setting we tackle a technical hurdle: Specifically, the graphs G_α associated with the nodes α of the trees $FT(u,v)$ are obtained from G by discarding $\leq f$ segments rather than edges, each of which may be long, and together these segments may contain much more than f edges. So we cannot apply Theorem 3.1 to the paths P_α which are indeed shortest paths but in graphs that may have obtained from G by discarding more than f edges.

It follows that in order to use Theorem 3.1 we change the definition of the paths P_α. A natural first attempt is to define P_α to be the shortest f-decomposable path between u and v in G_α. The path $P = P_{G\setminus F}(u,v)$ is f-decomposable, so as long as P survives in G_α it is a suitable candidate for being elected as P_α.

But, for P to survive as we go from a node $\alpha \in FT(u,v)$ to its child, we need that no edge $e_P \in P$ will be contained together with an edge of F in a segment of P_α. In section 2 we guaranteed that this is indeed the case as long as all edges of F are far from P in $G \setminus F$, specifically, outside of $\mathrm{tr}_{G\setminus F}(P)$. The argument was based on the fact that if some edge $e_P = (x,y)$ of P appears also on P_α (and assume it is closer to u than to v) then the length of $\mathrm{seg}(e, P_\alpha)$ is about $\epsilon|P_\alpha[u,x]|$ which is not larger than $\epsilon|P[u,x]|$ since P_α was a shortest path in G_α. Therefore this segment cannot reach an edge of F which must be outside $\mathrm{tr}_{G\setminus F}(P)$. When P_α is a shortest f-decomposable path then it could be that $P_\alpha[u,x]$ is much larger than $P[u,x]$. This could happen, for example, if $P_\alpha[x,v]$ is a concatenation of, say $f-2$ shortest path in G, but also contains an edge in F so it does not exist in $G \setminus F$. (We ignore for simplicity the interleaving edges in this example.) On the other hand $P[x,v]$ is a concatenation of $\delta \ll f$ shortest paths in G. This forces $P_\alpha[u,x]$ to be a concatenation of at most 2 shortest paths in G whereas $P[u,x]$ can be assembled from many more shortest paths and therefore be much shorter than $P_\alpha[u,x]$.

To overcome this difficulty we introduce the following additional type of paths.

DEFINITION 3.2. (k-EXPATH) *Let G be a graph and $G \setminus A$ a subgraph of G obtained by discarding the edges in A. A k-expath in $G \setminus A$ is path in $G \setminus A$ which is a concatenation of $(1 + 2\log D)$, k-decomposable paths in $G \setminus A$, such that the length of the i-th k-decomposable path is at most $\ell_i = \min\{2^i, 2^{2\log D - i}\}$.*

In the improved structure that we describe in this section, we choose P_α to be a shortest $(2f+1)$-expath from u to v in G_α. We prove that such a path has the following prefix optimality property. This property will allow us to prove that $P = P_{G\setminus F}(u,v)$ survives in the graphs G_α along the path that we traverse in $FT(u,v)$.

LEMMA 3.1. *Let $A \subset E$ and let P be a shortest k-expath from u to v in $G \setminus A$. Then for every $x \in V(P)$: $|P[u,x]| \leq 4|P^k_{G\setminus A}(u,x)|$ and $|P[x,v]| \leq 4|P^k_{G\setminus A}(x,v)|$.*

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 09/03/20 to 94.223.125.90. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Proof. Let P be a shortest k-expath from u to v in $G \setminus A$. Assume, by contradiction, that there is a vertex $x \in V(P)$ such that $|P[u,x]| > 4|P^k_{G\setminus A}(u,x)|$ or $|P[x,v]| > 4|P^k_{G\setminus A}(x,v)|$. Assume without loss of generality that $|P[u,x]| > 4|P^k_{G\setminus A}(u,x)|$ (the case where $|P[x,v]| > 4|P^k_{G\setminus A}(x,v)|$ is symmetric). We show that $P' = P^k_{G\setminus A}(u,x) \cdot P[x,v]$ is also a k-expath in $G \setminus A$, and since P' is shorter than P this contradicts the assumption that P is the shortest k-expath from u to v.

Since P is a k-expath in $G \setminus A$ then it is a concatenation of k-decomposable paths $P_0, P_1, \ldots, P_{2\log D}$ in $G \setminus A$, such that $|P_i| \le \ell_i = \min\{2^i, 2^{2\log D - i}\}$. To show that P' is a k-expath in $G \setminus A$ we define k-decomposable paths $P'_0, P'_1, \ldots, P'_{2\log D}$ in $G \setminus A$ such that $|P'_i| \le \ell_i$ and P' is the concatenation of $P'_0, P'_1, \ldots, P'_{2\log D}$.

Since $|P_i| \le \ell_i \le 2^i$ it follows that $\sum_{i=0}^{t} |P_i| < 2^{t+1}$. So if we let $j' = \lfloor \log |P[u,x]| \rfloor - 1$ then $\sum_{i=0}^{j'} |P_i| < |P[u,x]|$ and therefore x must be contained in P_j for some $j > j'$. Since clearly $j' < \log D$ we have that $\ell_{j'} = 2^{\lfloor \log |P[u,x]| \rfloor - 1} \ge |P[u,x]|/4 > |P^k_{G\setminus A}(x,u)|$.

We are now ready to define $P'_0, P'_1, \ldots, P'_{2\log D}$. We set $P'_i = \emptyset$ for every $0 \le i < j'$, $P'_{j'} = P^k_{G\setminus A}(u,x)$, $P'_i = \emptyset$ for every $j' < i < j$, P'_j to be the suffix of P_j starting from x, and $P'_i = P_i$ for every $i > j$.

It is easy to verify that for every i, P'_i is a k-decomposable path in $G \setminus A$ and that $|P'_i| \le \ell_i$. This proves that P' is a shorter k-expath from u to v than P in $G \setminus A$, which contradicts the assumption that P is the shortest k-expath from u to v.

As already mentioned the critical change that we make to the structure of section 2, in order to make it more space efficient, is to take P_α in each node $\alpha \in FT(u,v)$ to be a shortest $(2f+1)$-expath from u to v in G_α. We represent each such $(2f+1)$-expath, P_α, in a straightforward way, by a list of the endpoints of the paths, and the edges in the representation of each of its $\le 2\log D$ $(2f+1)$-decomposable paths. The size of this representation is $O(f \log D)$.

In addition, in order to be able to search the expaths efficiently, and later to report the shortest path from s to t in $G \setminus F$ if required, we store, as part of the data structure, all the shortest paths in G. Concretely, we store the values $\mathrm{d}_G(u,v)$, and $\pi(u,v)$, which is the predecessor of v on the shortest path from u to v, for every pair of vertices u and v. We assume that shortest paths in G are unique (which could be easily achieved with high probability, by slightly perturbing the edge weights).

The technical details of the construction and its analysis are given in the following sections, proving Theorem 1.1.

3.1 Reducing Space by Using Decomposable Paths - Technical Details.

We first make the following technical change. In Section 2 we used $\epsilon' = \epsilon/3$ to define both the trapezoid of a path P, and the netpoints of a path P. Here, we change the definition of the netpoints of a path P and use $\epsilon'' = \epsilon'/4$ there instead of ϵ', but keep ϵ' in the definition of a trapezoid. The definition of netpoints is now:

Definition 3.3. (Path Netpoints) *Let $P = (v_1 = u, v_2, \ldots, v_k = v)$ be a path from u to v in G. Let L be the set of all vertices $v_j, v_{j+1} \in P$ such that $|P[u,v_j]| < (1+\epsilon'')^i \le |P[u,v_{j+1}]|$ for some integer $i \ge 0$. Define R analogously to be the set of all vertices $v_j, v_{j+1} \in P$ such that $|P[v_{j+1},v]| < (1+\epsilon'')^i \le |P[v_j,v]|$ for some integer i. Let* $\mathrm{netpoints}(P) = L \cup R \cup \{u,v\}$.

This makes Lemma 2.1 hold with ϵ'' rather than ϵ'. Specifically, we have

Lemma 3.2. *Let P be a path from u to v, $e = (x,y)$ be an edge of P such that $|P[u,x]| < |P[u,y]|$. Then either* $\mathrm{seg}(e,P) = \{e\}$ *or* $|\mathrm{seg}(e,P)| \le \epsilon'' \min\{|P[u,x]|, |P[x,v]|\}$.

Clearly, the size of the data structure after these modifications is $O(n^2(\log D/\epsilon)^f \cdot f \log D)$. We now describe how to adapt the query algorithm to these modifications of the data structure.

Query. We implement a query as in Section 2 but we have to adapt the way we navigate through each tree $FT(u,v)$, to the new representation of the paths P_α. This navigation requires an algorithm, that given an edge e and an $(2f+1)$-expath P_α determines if $e \in P_\alpha$, and if so, returns the index of $\mathrm{seg}(e, P_\alpha)$ (this is the index of the child of α to which the search continues).

An edge $e = (x,y)$ is on the shortest path in G from u to v if and only if $\mathrm{d}_G(u,v) = \min\{\mathrm{d}_G(u,x) + w(x,y) + \mathrm{d}_G(y,v), \mathrm{d}_G(u,y) + w(x,y) + \mathrm{d}_G(x,v)\}$. To test if e is contained in an f-expath, P_α, we simply run the test above for every shortest path (in G) in the representation of P_α, and compare e to all the edges in the representation of P_α. If e is contained in one of the shortest paths in the representation of P_α, or if e itself belongs to this representation, then e is contained in P_α and otherwise it is not.

To find $\mathrm{seg}(e, P_\alpha)$ we compute $|P_\alpha[u,x]|$ and $|P_\alpha[x,v]|$ and $|P_\alpha|$. To compute $|P_\alpha[u,x]|$ we sum the lengths of the shortest paths and the edges, in the representation of P_α, between u and e, or the shortest path Q, in the representation of P_α, containing e. If e is contained in a shortest path Q in the representation of P_α, we also add $\mathrm{d}_G(w,x)$ where w is the endpoint closer to x of Q. The computation of $|P_\alpha[x,v]|$ is analogous. We

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 09/03/20 to 94.223.125.90. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

compute $|P_\alpha|$ by summing up the lengths of all the paths and edges in the representation of P_α. Clearly, computing $|P_\alpha[u,x]|$ and $|P_\alpha[x,v]|$ and $|P_\alpha|$ takes $O(f \log D)$ time.

Once we have $|P_\alpha[u,x]|$ and $|P_\alpha[x,v]|$ we compute the largest index i_1 such that $(1+\epsilon'')^{i_1} \leq |P_\alpha[u,x]|$. The number of netpoints in L that are between u and x on P_α (possibly including x) is $2i_1$. Similarly, we compute the largest index i_2 such that $(1+\epsilon'')^{i_2} \leq |P_\alpha[x,v]|$ and then deduce that $2i_2$ netpoints of R are between x and v on P_α. We also compute the total number of endpoints on P_α by computing the largest i such that $(1+\epsilon'')^i \leq |P_\alpha|$. It is easy to see how to find $\text{seg}(e, P_\alpha)$ from the indices i_1, i_2 and i.[2] We compute i_1, i_2, and i by a binary search of $\leq -ngthSP_\alpha[u,x]$, $|P_\alpha[x,v]|$, and $|P_\alpha|$, among the $\log_{1+\epsilon} D = (\log D)/\epsilon$ possible powers of $1+\epsilon''$.[3]

In total it takes us $O(f \log D + \log(\log D/\epsilon)) = O(f \log D + \log \frac{1}{\epsilon}) = O(f \log D)$ (note that the last equation holds since we assume $\epsilon > 1/D$) time to decide if $e \in P_\alpha$ and to find $\text{seg}(e, P_\alpha)$ if indeed $e \in P_\alpha$. Now recall that to perform a query we traverse $O(f^2)$ trees $FT(u,v)$, in each tree we traverse a path of length f, and in each node α we may have to scan through each edge of F to find one that is contained in P_α. For each such edge we apply the procedure just described. Therefore, the total query time is $O(f^5 \log D)$.

Preprocessing. For every pair of vertices u and v and for every node $\alpha \in FT(u,v)$ we have to compute the shortest $(2f+1)$-expath from u to v in G_α. We do this in three steps as follows.

1) We compute all pairs shortest paths in G_α and identify the pairs of vertices x and y for which $\text{d}_G(x,y) = \text{d}_{G_\alpha}(x,y)$. These are the pairs for which the shortest path in G_α is the same as the shortest path in G.

2) In this step we compute for every pair of vertices x and y the shortest $(2f+1)$-decomposable path in G_α. We form a layered directed graph G^2_α consisting of $2(2f+1)-1$ layers $V_1, V_2, \ldots, V_{2(2f+1)-1}$. Each layer contains a copy of each vertex of G. We denote the copy of a vertex x at layer i by x_i. The arcs of this layered graph are defined as follows. For $x \neq y$, we add an arc (x_1, y_2) if and only if $\text{d}_G(x,y) = \text{d}_{G_\alpha}(x,y)$. We set the weight of the arc (x_1, y_2) to be equal to $\text{d}_G(x,y)$. We put an arc (x_2, y_3) if there is an an edge (x,y) in G_α. We set the length of the arc (x_2, y_3) to be equal to the length of (x,y). We add arcs between subsequent levels so that for odd i, the arcs between V_i and V_{i+1} are the same as the arcs between V_1 and V_2, and for even i, the arcs between V_i and V_{i+1} are the same as the arcs between V_2 and V_3. I.e. if we have an arc (x_1, y_2) we also have an arc (x_i, y_{i+1}) of the same length for odd i, and if we have an arc (x_2, y_3) we also have an arc (x_i, y_{i+1}) of the same length for even i. We also put an arc (x_i, x_{i+1}) of length 0 for every x and i.

We compute the shortest path in G^2_α from every vertex in the first layer to every vertex in the last layer. It is easy to verify that the length of the shortest path from x_1 to $y_{2(2f+1)-1}$ in G^2_α is equal to the length of the shortest $(2f+1)$-decomposable path from x to y in G_α. So following this step we know $P^{2f+1}_{G_\alpha}(x,y)$ for every pair x, y in G_α.

3) In this step we also build a directed layered network G^3_α with $2\log D + 1$ layers $V_1, V_2, \ldots, V_{2\log D+1}$. Each layer contains a copy of every vertex of G. We denote the copy of a vertex x at level i by x_i. For $x \neq y$ we add an arc (x_i, y_{i+1}) of length $|P^{2f+1}_{G_\alpha}(x,y)|$ if $|P^{2f+1}_{G_\alpha}(x,y)| \leq \ell_i$. For every vertex u and index i we add an arc (x_i, x_{i+1}) of length 0.

Recall that we want to set P_α to be the shortest $(2f+1)$-expath from u to v in G_α. We compute the shortest path P^3 from u_1 to $v_{2\log D}$ in G^3_α. The length of P^3 is equal to the length of P_α. Let $P_1, P_2, \ldots, P_{2\log D}$ be the $(2f+1)$-decomposable paths composing P_α. The ith edge of P^3 corresponds to P_i. If this edge is of the form (x_i, x_{i+1}) then P_i is empty and if it is of the form (x_i, y_{i+1}) for $y \neq x$ then $P_i = P^{2f+1}_{G_\alpha}(x,y)$. Following these observations we extract the representation of P_α from G^2_α and G^3_α and store it at α.

The graph G^2_α has $O(fn^2)$ arcs so we can compute all the shortest paths in G^2_α from vertices of the first layer to vertices of the last layer in $O(fn^3)$ time (using Dijkstra's algorithm to compute shortest paths from every vertex in the first layer). The graph G^3_α has $O(n^2 \log D)$ arcs so we can compute the shortest path from u_1 to $v_{2\log D}$ in G^3_α in $O(n^2 \log D + n \log D \log n \log D) = O(n^2 \log D + n \log D \log\log D)$ time. It follows that the preprocessing time at each node α is $O(fn^3 + n^2 \log D + n \log D \log\log D)$ and by multiplying by the total number of nodes in all the trees $FT(u,v)$ we get that the total preprocessing time is $O((fn^5 + n^4 \log D + n^3 \log D \log\log D)(\log D/\epsilon)^f)$.

The following theorem summarizes the properties of the data structure that we have presented and proves its correctness.

THEOREM 3.2. *The f-sensitive distance oracle presented in this section requires $O(n^2(\log D/\epsilon)^f \cdot f \log D)$ space, can be built in $O((fn^5 + n^4 \log D + n^3 \log D \log\log D)(\log D/\epsilon)^f)$ time, and can answer a query in $O(f^5 \log D)$ time. Our distance oracle can also report an approximate shortest path from s to t in $G \setminus F$*

[2] Here we assume that we do not remove duplicate netpoints but introduce empty segments between them.

[3] These powers are independent of the query and can be stored with the data structure.

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 09/03/20 to 94.223.125.90. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

in time proportional to the length of this path.

Proof. The running times specified in the statement of theorem are clear from the preceding discussion. We are only left to prove the correctness of the data structure.

For correctness we follow the footsteps of the correctness proof in Section 2. A close inspection of the proof shows that the modifications which we did in the structure affects mainly Lemma 2.5.

In Section 2 we proved Lemma 2.5 for an arbitrary path P such that $\mathrm{tr}_{G\setminus F}(P) \cap V(H) = \emptyset$. However, when we established correctness in Theorem 2.1, we used Lemma 2.5 only for paths P which are shortest paths in $G \setminus F$ or concatenations of two shortest paths in $G \setminus F$. A shortest path in $G \setminus F$ is f-decomposable by Theorem 3.1. It follows immediately that a path which is a concatenation of two shortest paths in $G \setminus F$ is $(2f+1)$-decomposable. Therefore it suffices if we reprove Lemma 2.5 for $(2f+1)$-decomposable paths.

We established Lemma 2.5 in Section 2 by showing that P is contained in all graphs G_α, for nodes α along the path that the query algorithm follows in $FT(u,v)$. We showed this by induction on the nodes of this path starting from the root. We do the same here and indicate the modifications needed in the induction step using the same notation as in the proof of Lemma 2.5. Here we assume that P is a $(2f+1)$-decomposable path from u to v in $G \setminus F$ such that $\mathrm{tr}(P) \cap V(H) = \emptyset$.

The critical Equation there in the proof of Lemma 2.5 is Equation (2.1). This equation does not hold here since P_α is not a shortest path anymore. But since P_α is a shortest $(2f+1)$-expath then by Lemma 3.1 we have

$$\begin{aligned}(3.6)\qquad &\min\{|P_\alpha[u,x]|\,,|P_\alpha[x,v]|\} \le \\ &4\min\{|P^{2f+1}_{G\setminus F}(u,x)|, |P^{2f+1}_{G\setminus F}(x,v)|\}\end{aligned}$$

Continuing exactly the same reasoning as in the proof of Lemma 2.5 we get

$$\begin{aligned}&|\mathrm{seg}(e_F,P_\alpha)| \ge |P_\alpha[x,y]| \ge \mathrm{d}_{G\setminus F}(x,y)\\ (3.7)\qquad &> \epsilon' \min(|P(u,x)|\,,|P(x,v)|)\\ (3.8)\qquad &\ge \epsilon' \min\{|P^{2f+1}_{G\setminus F}(u,x)|, |P^{2f+1}_{G\setminus F}(x,v)|\\ (3.9)\qquad &\ge \frac{\epsilon'}{4}\min\{|P_\alpha[u,x]|\,,|P_\alpha[x,v]|\}\\ &= \epsilon''\min\{|P_\alpha[u,x]|\,,|P_\alpha[x,v]|\}\end{aligned}$$

where Equation (3.7) follows since $\mathrm{tr}(P) \cap V(H) = \emptyset$, Equation (3.8) follows since P is a $(2f+1)$-decomposable path, and Equation (3.9) follows by Equation (3.6).

However, since we modified our definition of net-points to use an exponential scale with base $1+\epsilon''$, it follows from Lemma 3.2 that

$$(3.10)\qquad |\mathrm{seg}(e_F,P_\alpha)| \le \epsilon'' \min\{|P_\alpha[u,x]|\,,|P_\alpha[x,v]|\}$$

which is a contradiction.

To report an approximate shortest path from s to t in $G \setminus F$ we follow $P_H(s,t)$. Each edge $e=(u,v)$ on $P_H(s,t)$ corresponds to some path P_α in $G \setminus F$ which is stored at node α of $FT(u,v)$. We report the concatenation of these paths P_α for all the edges along $P_H(s,t)$. Each such path P_α is by itself a concatenation of $O(f \log D)$ shortest paths and edges of G. To report each shortest path in the representation of P_α, we use the table π. Recall that for every pair of vertices u and v, $\pi(u,v)$ is the predecessor of v on the shortest path from u to v.

4 A Reduction From Arbitrary Weights To Bounded Weights.

In this section we reduce size of the data-structure by replacing the $\log^f D$ factor in the size complexity to $\log^f n$. We do this by a reduction from the problem of f-sensitive distance oracles on a graph with arbitrary edge weights to $\frac{\log W}{\log n}$ independent problems of f-sensitive distance oracles on graphs with bounded weights, where the minimum edge weight is 1 and the heaviest edge weight is $O(\mathrm{poly}(n))$.

LEMMA 4.1. *Assume we have an f-sensitive distance oracle with stretch α for bounded weights graphs (where the minimum edge weight is* 1 *and the heaviest edge weight $W = O(\mathrm{poly}(n))$) with $f_s(n)$ size, $f_q(n)$ query time and $f_p(n)$ preprocessing time. Then we can build an f-sensitive distance oracle for a graph with arbitrary edge weights whose size is $O(f_s(n)\cdot\frac{\log W}{\log n})$, with* $(1+o(1))\alpha$ *stretch, $O(f_q(n)\cdot\log\log W)$ query time and $O(f_p(n)\cdot\frac{\log W}{\log n}))$ preprocessing time.*

Proof. We first define graphs G^i and $\widetilde{G^i}$ for $1 \le i \le \frac{\log W}{\log n}$, which we derive from G using different rounding scales. We show how to find the index i such that the shortest s to t path in G^i has $(1+o(1))\alpha$ stretch compared to the shortest s to t path in $G \setminus F$.

Define G^i to be the following graph. Replace edge weights $\le n^{i-2}$ by n^{i-2}. Discard all edges of weight $\ge n^{i+1}$. We show the following monotonicity property on the graphs G^i. There is some index i such that s and t are disconnected in $G^{i'}$ for every $i' \le i-2$, while s and t are connected in $G^{i'}$ for every $i' \ge i$.

Let $W_{(F,s,t)}$ be the weight of the heaviest edge on the shortest path from s to t in the graph $G \setminus F$. Let $i = \lfloor \log_n W_{(F,s,t)} \rfloor$, then $n^i \le W_{(F,s,t)} < n^{i+1}$. Note that this implies $\mathrm{d}_{G\setminus F}(s,t) \ge n^i$.

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 09/03/20 to 94.223.125.90. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

First we show that s and t are disconnected in $G^{i'} \setminus F$ for every $i' \leq i-2$. To see this, note that the maximal edge weight in $G^{i'}$ is less than $n^{i'+1} \leq n^{i-1}$. hence, if s and t are connected in $G^{i'} \setminus F$ then they are connected by a path length less than n^i, contradiction to the fact that $\mathrm{d}_{G \setminus F}(s,t) \geq n^i$.

Next, we show that s and t are connected in $G^{i'} \setminus F$ for every $i' \geq i$. Consider $i' \geq i$. Recall that $G^{i'}$ contains all edges whose weight is $< n^{i'+1}$ — in particular $G^{i'}$ contains all edges whose weight is $< n^{i+1}$. It follows that $G^{i'} \setminus F$, contains all edges of $G \setminus F$ whose weight is $< n^{i+1}$. Hence, $G^{i'} \setminus F$ contains all edges along the shortest path from s to t in $G \setminus F$. Thus, s and t are connected in $G^{i'} \setminus F$.

We now show that querying (F,s,t) in G^i returns a $(1+o(1))\alpha$ approximate shortest path in the graph $G \setminus F$. We first show that $\mathrm{d}_{G^i \setminus F}(s,t) \leq (1+1/n)\mathrm{d}_{G \setminus F}(s,t)$. Recall that $\mathrm{d}_{G \setminus F}(s,t) \geq n^i$. Rounding up all edges of weight less than n^{i-2} to n^{i-2} increases the shortest path by at most $(n-1) \cdot n^{i-2} < n^{i-1}$. Hence $\mathrm{d}_{G^i \setminus F}(s,t) \leq \mathrm{d}_{G \setminus F}(s,t) + n^{i-1} \leq (1+1/n)\mathrm{d}_{G \setminus F}(s,t)$. Recall that the stretch of D^i is α which means it returns an α-approximation of $\mathrm{d}_{G^i \setminus F}(s,t)$. It follows that the length we return $\leq \alpha \mathrm{d}_{G^i \setminus F}(s,t) \leq (1+1/n)\alpha \mathrm{d}_{G \setminus F}(s,t) = (1+o(1))\alpha \mathrm{d}_{G \setminus F}(s,t)$.

The graph G_i has minimum edge weight n^{i-2} and maximum edge weight n^{i+1}, we want to scale it so that the minimum edge weigh will be 1. Divide all edge weights of G^i by n^{i-2} and round upwards to the closest integer, call the resulting graph $\widetilde{G^i}$. Obviously, in $\widetilde{G^i}$, the minimum edge weight is 1 and maximum edge weight is n^3. We now show that $n^{i-2} \cdot \mathrm{d}_{\widetilde{G^i} \setminus F}(s,t) \leq (1+1/n)\mathrm{d}_{G^i \setminus F}(s,t)$. The length of any simple s to t path in G^i is at least n^i, and if we round the edge weights upwards to the closest multiple of n^{i-2} we add to it a length of at most $(n-1) \cdot n^{i-2} < n^{i-1}$. Thus, the length of the path in $\widetilde{G^i}$ multiplied by n^{i-2} is at most $(1+1/n)$ the length in G^i. We conclude that $n^{i-2} \cdot \mathrm{d}_{\widetilde{G^i} \setminus F}(s,t) \leq (1+1/n)\mathrm{d}_{G^i \setminus F}(s,t) \leq (1+o(1))\alpha \mathrm{d}_{G \setminus F}(s,t)$, where in the last inequality we used that $\mathrm{d}_{G^i \setminus F}(s,t) \leq (1+o(1))\alpha \mathrm{d}_{G \setminus F}(s,t)$.

So far, we have described the graphs G^i and $\widetilde{G^i}$. We now describe the data-structure we build. In the preprocessing stage we compute all $G^i, \widetilde{G^i}$ for all $1 \leq i \leq \frac{\log W}{\log n}$, and store the an f-sensitive distance oracle D_i for every $\widetilde{G^i}$. Thus, space requirements are $O(f_s(n) \cdot \frac{\log W}{\log n})$ and preprocessing time is $O(f_p(n) \cdot \frac{\log W}{\log n}))$. We now describe the procedure for answering query (F,s,t). We first do a binary search in the range $1 \leq i \leq \frac{\log W}{\log n}$, to find the minimal value i such that s and t are connected in G^i (querying the distance oracle D_i with (F,s,t) answers if s and t are connected or not in G^i). Then, using the distance oracles D_i and D_{i+1} we query (F,s,t) and multiply the result with n^{i-2} and n^{i-1}, respectively. We return the minimum of both alternatives. The claims above prove that the stretch is $(1+o(1))\alpha$. Due to the binary search, total query time is $O(f_q(n) \cdot \log\log W)$.

Observe that when we substitute $D = \mathrm{poly}(n)$ in Theorem 1.1, we obtain an f-sensitive distance oracle with stretch $\alpha = 1+\epsilon$ with the following properties:

- space complexity: $f_s(n) = O(n^2(\log n/\epsilon)^f \cdot f \log n)$.
- query time: $f_q(n) = O(f^5 \log n)$.
- preprocessing time: $f_p(n) = O(fn^5(\log n/\epsilon)^f)$.

By substituting these functions $f_s(n), f_q(n), f_p(n)$ in Lemma 4.1, we obtain the following Theorem.

Theorem 4.1. *Let G be a weighted undirected graph. There exists f-sensitive distance oracle for G with $(1+\epsilon)$ stretch, $O(n^2(\log n/\epsilon)^f \cdot f \log W)$ space, and $O(f^5 \log n \log\log W)$ query time. The oracle can be built in $O(fn^5(\log n/\epsilon)^f \frac{\log W}{\log n})$ time.*

5 Conclusion.

In this paper, we present the first f-sensitive distance oracle with $(1+\epsilon)$-stretch with small space and fast query time that handles multiple edge failures. For any $f = o(\frac{\log n}{\log\log n})$ and fixed $\epsilon > 0$ our f-sensitive distance oracle has $n^{2+o(1)}$ space, $\widetilde{O}(1)$ query time and $(1+\epsilon)$ stretch.

This is a big step forward in our understanding of this problem. We leave however many open questions. Firstly, our data-structure has large space requirements when $f >> \frac{\log n}{\log\log n}$ which are dominated by the factor of $O((\frac{\log n}{\epsilon})^f)$ in the space complexity. Can the space be reduced? Secondly, our data-structure deals with at most f edge-failures, what can be done in the case of vertex failures? Thirdly, can one develop efficient distance-sensitive oracles in the case of directed graphs for multiple failures? Another interesting direction is: can we develop efficient oracles for the seemingly easier problem of reachability ("is there some path from u to v after the failures?") for multiple edge/vertex failures?

References

[1] Ittai Abraham, Shiri Chechik, and Kunal Talwar. Fully dynamic all-pairs shortest paths: Breaking the $o(n)$ barrier. In *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 1–16, 2014.

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 09/03/20 to 94.223.125.90. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

[2] Yehuda Afek, Anat Bremler-Barr, Haim Kaplan, Edith Cohen, and Michael Merritt. Restoration by path concatenation: fast recovery of MPLS paths. *Distributed Computing*, 15(4):273–283, 2002.

[3] Giorgio Ausiello, Giuseppe F. Italiano, Alberto Marchetti-Spaccamela, and Umberto Nanni. Incremental algorithms for minimal length paths. *J. Algorithms*, 12(4):615–638, 1991.

[4] Surender Baswana, Sumeet Khurana, and Soumojit Sarkar. Fully dynamic randomized algorithms for graph spanners. *ACM Transactions on Algorithms*, 8(4):35, 2012.

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

[6] Aaron Bernstein and David Karger. Improved distance sensitivity oracles via random sampling. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 34–43, 2008.

[7] Aaron Bernstein and David Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC, pages 101–110, 2009.

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

[9] Camil Demetrescu and Giuseppe F. Italiano. Fully dynamic all pairs shortest paths with real edge weights. *J. Comput. Syst. Sci.*, 72(5):813–837, 2006.

[10] Camil Demetrescu and Mikkel Thorup. Oracles for distances avoiding a link-failure. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 838–843, 2002.

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

[12] Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 506–515, 2009.

[13] Shimon Even and Yossi Shiloach. An on-line edge-deletion problem. *J. ACM*, 28(1):1–4, 1981.

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

[15] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Dynamic approximate all-pairs shortest paths: Breaking the o(mn) barrier and derandomization. In *Proceedings of the 54th Annual Symposium on Foundations of Computer Science*, FOCS, pages 538–547, 2013.

[16] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Sublinear-time maintenance of breadth-first spanning tree in partially dynamic networks. In *Proceedings of the 40th International Colloquium, ICALP*, pages 607–619, 2013.

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

[18] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A subquadratic-time algorithm for decremental single-source shortest paths. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 1053–1072, 2014.

[19] Neelesh Khanna and Surender Baswana. Approximate shortest paths avoiding a failed vertex: Optimal size data structures for unweighted graphs. In *27th International Symposium on Theoretical Aspects of Computer Science, STACS*, pages 513–524, 2010.

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

[21] Valerie King and Mikkel Thorup. A space saving trick for directed dynamic transitive closure and shortest path algorithms. In *COCOON*, pages 268–277, 2001.

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

[23] Liam Roditty and Uri Zwick. Dynamic approximate all-pairs shortest paths in undirected graphs. *SIAM J. Comput.*, 41(3):670–683, 2012.

[24] Mikkel Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In *SWAT*, pages 384–396, 2004.

[25] Mikkel Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing (STOC)*, pages 112–119, 2005.

[26] Oren Weimann and Raphael Yuster. Replacement paths and distance sensitivity oracles via fast matrix multiplication. *ACM Trans. Algorithms*, 9(2):14:1–14:13, March 2013. Announced at FOCS'10.

[27] Ran Duan. Algorithms and dynamic data structures for basic graph optimization problems. PhD Thesis, University of Michigan, 2011.

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 09/03/20 to 94.223.125.90. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

A A Simple But Space Inefficient Construction.

We present the following construction in an attempt to make the construction in the body of the paper more accessible. This data-structure can accommodate up to f edge failures and returns the exact shortest path in the graph without the failed edges. Unfortunately, the space requirements are quite large, $O(n^{f+3})$. It may be helpful to consider Figure 8 when reading the description of the following data structure and query algorithm:

- for every pair of vertices $u, v \in V$, construct a tree BruteForce(u, v), of height at most f and every node has at most $n - 1$ children.
  - The root r stores the shortest path from u to v in G.
  - Let α be an arbitrary node of the tree, and let G_α be a subgraph of G associated with the node α (we define G_α recursively below).
  - Let P_α be the shortest path between u and v in G_α (denote by $P_\alpha = \emptyset$ if u and v are in different connected components of G_α). Store the path P_α in node α.
  - If $P_\alpha = \emptyset$ or α is at depth f then α is a leaf of BruteForce(u, v).
  - If the depth of $\alpha < f$ and $P_\alpha \neq \emptyset$, then we create a child node α' for every edge $e \in P_\alpha$. We associate the edge e with the tree arc from α to its child α' and denote it by $S_{\alpha'} = \{e\}$.
  - We now recursively define avoid(α), a set of edges we discard from G at node α. For the root r, avoid$(r) = \emptyset$. For a node α' which is a child of α, we define avoid$(\alpha') =$ avoid$(\alpha) \cup S_{\alpha'}$. In other words, avoid$(\alpha)$ is the set of edges associated with the arcs along the tree path from the root to α.
  - We define $G_\alpha = G \backslash \text{avoid}(\alpha)$, that is the graph G after all edges in avoid(α) are discarded.
- BruteForce(u, v) is a tree of height at most f, with degree at most n, and hence it contains $O(n^f)$ nodes. In each node α we store the path P_α using $O(n)$ space per node. Thus, each tree occupies $O(n^3)$ space. There are $O(n^2)$ such trees (one for each pair of vertices $u, v \in V$), so total space is $O(n^{f+3})$.
- To answer the query (F, s, t) we search the tree BruteForce(s, t) using the set of failed edges F as follows: We start with the root of BruteForce(s, t), if the path stored in the root does not contain any of the edges F — the answer to the query is the path stored in the root. Otherwise, we branch to a child node using an arbitrary edge in the intersection of F and the path stored at the root of BruteForce(s, t). We continue this search procedure until we either reach a leaf where s and t are disconnected (in which case we reply that s and t are disconnected in the graph G after all edges in F fail) or until we reach a node containing a path which is edge disjoint from F. This path is a shortest path from s to t in the graph G after all edges in F fail. The complexity of such a query is $O(f^2)$ since the query follows a downwards path from the root of BruteForce(s, t), (*i.e.*, of length $\leq f$), and we can check the intersection of F with the path stored in any node in time $O(f)$ using a hash table.

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 09/03/20 to 94.223.125.90. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

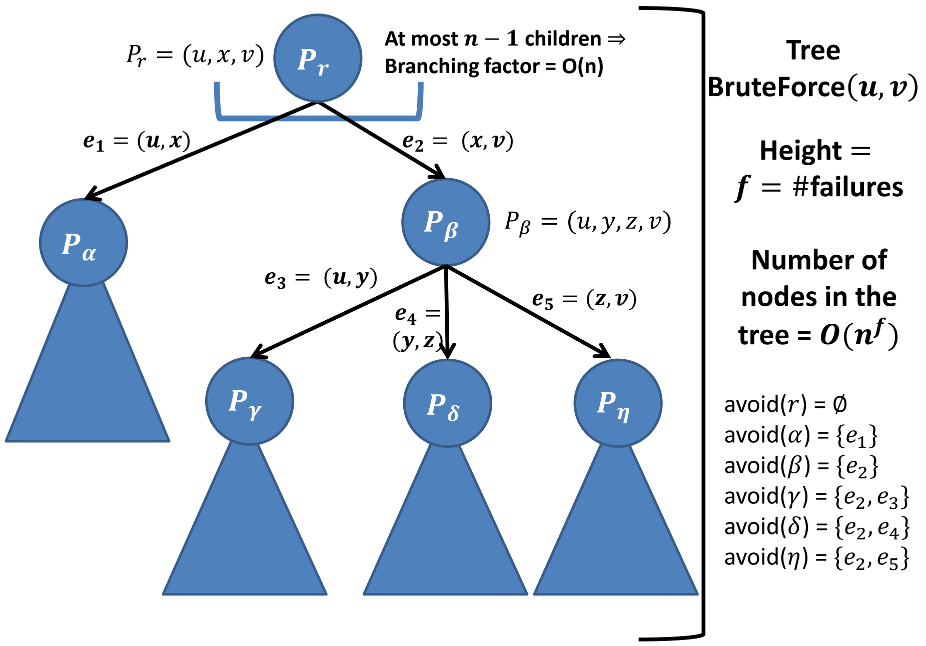


Figure 8: Illustration of the simple and exact $O(n^{f+3})$ space algorithm. Every node has at most $n-1$ children, one for each edge in the path stored within the node. Node β is a child of node r, the path associated with node r is the path $u \to x \to v$, the shortest path from u to v in the graph G. The path associated with node β, $u \to y \to z \to v$, is the shortest path from u to v after the edge (x, v) is deleted from G. The path associated with node γ is the shortest path from u to v after both edges (x, v) and (u, y) have been deleted from G, if such a path exists, or $\emptyset$ if there is no path from u to v in this graph.

Copyright © by SIAM
Unauthorized reproduction of this article is prohibited