Optimal Vertex Fault-Tolerant Spanners in Polynomial Time

Greg Bodwin[*]
University of Michigan
bodwin@umich.edu

Michael Dinitz[†]
Johns Hopkins University
mdinitz@cs.jhu.edu

Caleb Robelle
UMBC
carobel1@umbc.edu

Abstract

Recent work has pinned down the existentially optimal size bounds for vertex fault-tolerant spanners: for any positive integer k, every n-node graph has a $(2k-1)$-spanner on $O(f^{1-1/k}n^{1+1/k})$ edges resilient to f vertex faults, and there are examples of input graphs on which this bound cannot be improved. However, these proofs work by analyzing the output spanner of a certain exponential-time greedy algorithm. In this work, we give the first algorithm that produces vertex fault tolerant spanners of optimal size and which runs in polynomial time. Specifically, we give a randomized algorithm which takes $\widetilde{O}\left(f^{1-1/k}n^{2+1/k} + mf^2\right)$ time. We also derandomize our algorithm to give a deterministic algorithm with similar bounds. This reflects an exponential improvement in runtime over [Bodwin-Patel PODC '19], the only previously known algorithm for constructing optimal vertex fault-tolerant spanners.

1 Introduction

Let $G = (V, E)$ be a graph, possibly with edge lengths $w : E \to \mathbb{R}_{\geq 0}$. A t-spanner of G, for $t \geq 1$, is a subgraph $H = (V, E')$ that preserves all pairwise distances within a factor of t, i.e.,

$$(1.1) \qquad d_H(u, v) \leq t \cdot d_G(u, v)$$

for all $u, v \in V$ (where d_X denotes the shortest-path distance in a graph X). Since H is a subgraph of G it is also true that $d_G(u, v) \leq d_H(u, v)$, and so distances in H are the same as in G up to a factor of t. The distance preservation factor t is called the *stretch* of the spanner. Spanners were introduced by Peleg and Ullman [PU89a] and Peleg and Schäffer [PS89], and have a wide range of applications in routing [PU89b], synchronizers [AP90], broadcasting [ABP91, Pel00], distance oracles [TZ05], graph sparsifiers [KP12], preconditioning of linear systems [EEST08], etc.

The most common objective in spanners research is to achieve the best possible existential size-stretch trade-off, and to do this with algorithms that are as fast as possible. Most notably, a landmark result of Althöfer et al. [ADD+93] analyzed the following simple and natural greedy algorithm: given an n-node graph G and an integer $k \geq 1$, consider the edges of G in non-decreasing order of their weight and add an edge (u, v) to the current spanner H if and only if $d_H(u, v) > (2k-1)w(u, v)$. They proved that this algorithm produces $(2k-1)$-spanners of *existentially optimal size*: the spanner produced has size $O(n^{1+1/k})$, and (assuming the well-known *Erdős girth conjecture* [Erd64]) there are graphs in which *every* $(2k-1)$ spanner (and in fact every $2k$-spanner) has at least $\Omega(n^{1+1/k})$ edges.

1.1 Fault Tolerance

A crucial aspect of real-life systems that is not captured by the standard notion of spanners is the possibility of *failure*. If some edges (e.g., communication links) or vertices (e.g., computer processors) fail, what remains of the spanner might not still approximate the distances of what remains of the original graph. This motivates the notion of *fault tolerant* spanners:

DEFINITION 1.1. *A subgraph H is an f-vertex fault tolerant (f-VFT) t-spanner of G if*

$$(1.2) \qquad d_{H \setminus F}(u, v) \leq t \cdot d_{G \setminus F}(u, v)$$

for all $u, v \in V$ and $F \subseteq V \setminus \{u, v\}$ with $|F| \leq f$.

In other words, an f-VFT spanner contains a spanner of $G \setminus F$ for every set of $|F| \leq f$ nodes that could fail. The definition for edge fault tolerance (EFT) is equivalent, with the only change being that $F \subseteq E$ rather than $F \subseteq V \setminus \{u, v\}$.

Fault tolerant spanners were originally introduced in the geometric setting (where the vertices are points in $\mathbb{R}^d$ and the initial graph G is the complete graph with Euclidean distances) by Levcopoulos, Narasimhan, and Smid [LNS98a] and have since been studied extensively in that setting [LNS98b, Luk99, CZ04, NS07]. Chechik, Langberg, Peleg and Roditty [CLPR10] were the first to study fault-tolerant spanners in general graphs, giving a construction of an f-VFT $(2k-1)$-spanner of size

approximately $O(f^2 k^{f+1} \cdot n^{1+1/k} \log^{1-1/k} n)$ and an f-EFT $(2k-1)$-spanner of size $O(f \cdot n^{1+1/k})$. So they showed that introducing tolerance to f edge faults costs us an extra factor of f in the size of the spanner, while introducing tolerance to f vertex faults costs us a factor of $f^2 k^{f+1}$ in the size (compared to the size of a non-fault tolerant spanner of the same stretch).

Since [CLPR10], there has been a line of work focused on improving these bounds, particularly for vertex faults (see Table 1). The first improvement was by [DK11], who improved the bound for vertex faults to $O(f^{2-\frac{1}{k}} n^{1+\frac{1}{k}} \log n)$ via a black-box reduction to non-fault tolerant spanners. Following this, the area turned towards analyses of the *FT-greedy algorithm*, the obvious extension of the greedy algorithm of [ADD+93] to the fault tolerant setting: look at the edges (u, v) of the input graph G in order of nondecreasing weight, and add (u, v) to the spanner H iff currently there exists a set of $|F| \leq f$ faults such that (1.2) fails. This algorithm was first analyzed by [BDPW18] who obtained a size bound of $O(\exp(k) f^{1-1/k} n^{1+1/k})$. They also proved a lower bound of $\Omega(f^{1-1/k} n^{1+1/k})$ for f-VFT $(2k-1)$-spanners: assuming the girth conjecture of Erdös [Erd64], there are graphs which require that many edges for any f-VFT $(2k-1)$-spanner. An improved analysis of the FT-greedy algorithm was then given by [BP19], who remove the $\exp(k)$ factor and so proved that this algorithm gives existentially optimal VFT spanners.

While the FT-greedy algorithm inherits some of the nice properties of the non-faulty greedy algorithm (such as simplicity, easy-to-prove correctness, and existential optimality), it unfortunately has serious issues in runtime. The edge test in the FT greedy algorithm, i.e., whether or not there exists a fault set under which (1.2) holds, is an NP-hard problem known as LENGTH-BOUNDED CUT [BEH+06], and hence the algorithm inherently runs in exponential time. Addressing this, a *greedy algorithm with slack* was recently proposed in [DR20]. This algorithm is an adaptation of the FT-greedy algorithm which replaces the exponential-time edge test with a different subroutine $\text{TEST}(u, v)$, which accepts every edge (u, v) where there exist $|F| \leq f$ faults under which (1.2) fails, *and possibly some other edges too*. This slack maintains correctness and allows one to escape NP-hardness, but it introduces the challenge of bounding the number of additional edges added. The approach in [DR20] is to design an $O(k)$-approximation algorithm for LENGTH-BOUNDED CUT and use this in an efficiently computable TEST subroutine. This gives a polynomial runtime, but pays the approximation ratio of $O(k)$ in spanner size over optimal. So the result in [DR20] takes an important step forward (polynomial

time) but also a step back (non-optimal size, by a factor of $O(k)$).

It thus remains an important open problem to design a polynomial time algorithm which obtains truly optimal size. We note that k factors are often considered particularly important for spanners, since the regime $k = \Theta(\log n)$ yields the sparsest possible spanners and hence arises commonly in algorithmic applications (see, e.g., [BBG+20] for a recent example), and here an extra factor of k in the size of the spanner is significant. Accordingly, for spanners and many related objects there has been significant effort expended to remove unnecessary factors of k. It seems to often be the case that initial algorithms pay a factor of k, which can later be removed through more careful algorithms and analyses. Our results fit into this tradition, addressing the remaining open question: can we get truly optimal-size fault tolerant spanners in polynomial time?

1.2 Our Results and Techniques We answer this in the affirmative, giving both randomized and deterministic algorithms for constructing optimal-size fault-tolerant spanners. More formally, we prove the following theorems.

THEOREM 1.1. *There is a randomized algorithm which runs in expected time*

$$\widetilde{O}\left(f^{1-1/k} n^{2+1/k} + m f^2\right)$$

which with high probability returns an f-VFT $(2k-1)$-spanner with $O(f^{1-1/k} n^{1+1/k})$ edges.

THEOREM 1.2. *There is a deterministic algorithm which constructs an f-VFT $(2k-1)$ spanner with at most $O(f^{1-1/k} n^{1+1/k})$ edges in time*

$$\tilde{O}\left(f^{4-1/k} n^{2+1/k} + m f^5\right).$$

If $f = poly(n)$ (i.e., $f \geq n^c$ for some constant $c > 0$) then the running time improves to

$$\tilde{O}\left(f^{1-1/k} n^{2+1/k} + m f^2\right),$$

where the polynomial exponent c appears in both the spanner size and the running time but is hidden by the $O(\cdot)$ notation.

So if f is subpolynomial in n then our deterministic algorithm is slower than our randomized algorithm by about f^3, while if f is polynomial in n then we get determinism essentially for free (although the hidden polylogarithmic factors are larger in the deterministic case). Recent work by Karthik and Parter [CP21]

Spanner Size	Runtime	Greedy?	Citation
$\widetilde{O}\left(k^{O(f)} \cdot n^{1+1/k}\right)$	$\widetilde{O}\left(k^{O(f)} \cdot n^{3+1/k}\right)$		[CLPR10]
$\widetilde{O}\left(f^{2-1/k} \cdot n^{1+1/k}\right)$	$\widetilde{O}\left(f^{2-2/k} \cdot mn^{1+1/k}\right)$		[DK11]
$O\left(\exp(k)f^{1-1/k} \cdot n^{1+1/k}\right)$	$O\left(\exp(k) \cdot mn^{O(f)}\right)$	$\checkmark$	[BDPW18]
$O\left(f^{1-1/k} \cdot n^{1+1/k}\right)$	$O\left(mn^{O(f)}\right)$	$\checkmark$	[BP19]
$O\left(kf^{1-1/k} \cdot n^{1+1/k}\right)$	$\widetilde{O}\left(f^{2-1/k} \cdot mn^{1+1/k}\right)$	$(\checkmark)$	[DR20]
$O\left(f^{1-1/k} \cdot n^{1+1/k}\right)$	$\widetilde{O}\left(f^{1-1/k}n^{2+1/k} + mf^2\right)$	$(\checkmark)$	**(this paper)**

Table 1: Prior work on f-VFT $(2k-1)$-spanners of weighted input graphs on n nodes and m edges. Size bounds in red are existentially optimal, and runtimes in blue are polynomial. The ($\checkmark$) entries indicate a greedy algorithm with slack, as discussed below. With $\widetilde{O}$ we hide factors of $\log n$ (or k, since we may assume $k \leq \log n$).

provides a better derandomization which gives the same $\widetilde{O}\left(f^{1-1/k}n^{2+1/k} + mf^2\right)$ bound for all values of f.

To put these results in context, note that this is an *exponential* improvement in running time over [BP19], the only previous algorithm to give optimal-size fault tolerant spanners. And unlike [DR20, DK11] it gives spanners with existentially optimal size, saving an $O(k)$ factor over [DR20] and an $O(f \log n)$ factor over [DK11]. It is also polynomially faster than both [DR20, DK11]. We note that not only is $k = \Theta(\log n)$ (and superconstant k more generally) a particularly interesting regime (as discussed), large values of f are also particularly interesting. If we only ever think of f as small then the dependence on f in the size of the spanner does not matter much, but of course we are interested in protecting against as many faults as possible! So our results are strongest (compared to previous work) precisely in one of the most interesting regimes for fault-tolerant spanners: $k = \Theta(\log n)$ and f polynomially large in n.

Additional Properties. Our algorithms and techniques have a few other properties that we briefly mention here, but which will not be a focus in the paper. First, a corollary of our techniques and analysis is that we actually speed up the running time of the *non-fault tolerant* greedy algorithm from $O(mn^{1+1/k})$ to $O(kn^{2+1/k})$. Second, our algorithms and bounds continue to hold for edge fault tolerance, but for simplicity we will only discuss the VFT case. These size bounds are also not known to be optimal for edge fault tolerance (except for spanners of stretch 3) since the known lower bounds are weaker, making our results more interesting in the VFT setting. See Section 6 for more discussion of edge fault tolerance. Finally, since our algorithms are slack-greedy, they are *unconditionally optimal*: even if the Erdős girth conjecture is false, our algorithms still

produce spanners of optimal size (whatever that is).

1.2.1 Our First Algorithm Our first algorithm is a surprisingly simple randomized algorithm that, while not as efficient as the algorithm we will use to prove Theorem 1.1, achieves our main goal: it has polynomial running time and produces spanners of optimal size. It also illustrates the main ideas that our more advanced algorithms (faster and/or deterministic) will utilize.

THEOREM 1.3. *There is a randomized algorithm which, given an undirected weighted n-node graph and positive integers f and k, runs in polynomial time and with high probability returns an f-VFT $(2k-1)$-spanner with $O(f^{1-1/k}n^{1+1/k})$ edges.*

The main new ingredient is the following simple TEST(u,v) subroutine. To test an edge (u,v) we first randomly sample $\Theta(\log n)$ induced subgraphs of the current spanner, each of which is obtained by including u and v and then including each other node with probability $1/(2f)$. Then we test (1.1) in each of the sampled subgraphs, and we add (u,v) to the spanner iff a large enough fraction of these subgraphs violate (1.1). For correctness, one observes that if there exists any set of $|F| = f$ vertex deletions under which $d_{H \setminus F}(u,v)$ is large, then with high probability a large fraction of the subgraphs will delete all of F. Hence $d(u,v)$ will be large in these subgraphs, and we will correctly include (u,v) in the spanner.

The more interesting part of the argument is bounding the size of the output spanner. This relates to the *blocking set technique* introduced in [BP19] and also used in [DR20]. This technique uses the following observation: if the spanner H is sufficiently dense, and one samples a random induced subgraph on n/f nodes, then that subgraph will still be dense enough that it

must have some $\leq 2k$-cycles. This statement is even somewhat robust, in that one will not be able to destroy all $\leq 2k$-cycles by deleting only a constant fraction of remaining edges in the subgraph. Thus one can certify *sparsity* of the spanner by arguing that it is in fact possible to sample a subgraph, remove a constant fraction of the surviving edges, and destroy all $\leq 2k$-cycles in the process. In [BP19, DR20], one roughly predefines a small set of edges in the spanner (called the "blocking set") that intersects all $\leq 2k$-cycles; the strategy is then to sample a subgraph and remove the parts of the blocking set that survive. This approach works, but it is limited: if we use a slack FT-greedy algorithm (as in [DR20]) then the size of the blocking set increases with the slack, giving spanners that no longer have optimal size.

Our main idea is to bypass blocking sets by more closely tying together the algorithm and the analysis. The analysis of the blocking set technique uses random subgraph sampling, but this does not appear in the actual *algorithm* of [DR20] or [BP19]. Our new algorithm, by explicitly sampling subgraphs as part of the TEST(u, v) subroutine, is in some sense doing algorithmically exactly the *minimum* needed for the analysis to work. As is shown more formally in Section 3, if an edge passes our new TEST(u, v) then *by construction* the probability that it will have to be removed in the analysis in order to obtain high-girth is at most some constant less than 1. So getting our sampled subgraph to be high-girth in the analysis requires removing only a constant fraction of the remaining edges. This is in contrast to the ideas behind blocking sets, where the blocking set is predefined by the algorithm and so an edge either has to be removed from the analysis subgraph (if it is part of the blocking set) or does not (if it isn't). Hence another way to think of our idea is that we are moving from a *global* analysis of which edges need to be removed (blocking sets, where edges are either in the set or not) to a *local* analysis (where each edge has only a constant probability of being removed).

Details of this algorithm and analysis are given in Section 3.

1.2.2 Our Faster Randomized Algorithm As discussed, obtaining *fast* algorithms for spanners – not just any polynomial time – is a long-standing and important line of research. Once we are able to achieve polynomial time, we naturally want to minimize this time. Our next algorithm (which achieves the running time bound of Theorem 1.1) optimizes the runtime in two ways:

1. It is costly to randomly sample subgraphs anew in each round of the algorithm. A more efficient approach is to randomly sample vertex sets once at the very beginning, use these to perform the TEST in each round, and incrementally maintain the sampled subgraphs as edges are added to the spanner. We show that this approach works as long as we sample $\Theta(f^3 \log n)$ total subgraphs in the beginning.

2. Since we incrementally maintain a fixed set of subgraphs, to test (1.1) on each subgraph we can use an appropriate incremental data structure rather than computing from scratch each time. The obvious way of doing this requires using an incremental dynamic distance oracle / APSP algorithm, but unfortunately known constructions are not fast enough for our purposes. However, we can use an idea from [RZ11]: it suffices to solve a certain relaxed version of this problem. Specifically, instead of measuring $d(u, v)$ exactly; it suffices for our oracle to simply decide whether $d(u, v) > (2k - 1)w(u, v)$ in each subgraph. In the setting of unweighted input graphs, this easily reduces to a problem of *reachability* (rather than distance) on a $(k + 1)$-layered version of the subgraph, and we can use an observation from [DR20] to make this reduction work for weighted input graphs as well thanks to the fact that our framework is a slack version of the greedy algorithm. We can then use a classical data structure for incremental reachability by Italiano [Ita86].

Together, these improvements give the runtime listed in Theorem 1.1. Details are given in Section 4.

1.2.3 Our Deterministic Algorithm Our second improvement is to regain determinism. Both the exponential time algorithm of [BDPW18] and the polynomial time algorithm of [DR20] are deterministic, while the core ideas of our previous two algorithms seem to require randomization (particularly our first, non-optimized algorithm). But by constructing set systems with specific properties through the use of almost-universal hash functions, we are able to derandomize the algorithm of Theorem 1.1.

The main idea is to leverage the fact that our fast randomized algorithm samples vertex sets only once at the very beginning. By examining the proof of Theorem 1.1 we can determine what properties we need these sets to have. Informally, we need that for every $(u, v) \in E$ there are not many sets containing both u and v, and that for every $(u, v) \in E$ and $F \subseteq V \setminus \{u, v\}$ with $|F| \leq f$, a constant fraction of the sets which contain both u and v do not contain any vertex in F (note that this guarantee has to hold

simultaneously for all possible fault sets). So we just need to give a deterministic construction of such a set system. We show how to do this by building an (almost-)universal hash family from V to $[\Theta(f)]$, and for each hash function in the family creating $\Theta(f^2)$ sets based on pairs of hash values. Interestingly, we are able to tolerate relatively large values of "almost": our spanner construction still works even if the universal hashing guarantee is violated by large constants. Most of the literature on hashing, on the other hand, is optimized for the case of only a $(1+\epsilon)$ violation. By taking advantage of our ability to withstand weaker hashing guarantees, we can design an extremely small hash family based on *Message Authentication Codes* from cryptography, which is a standard and classical construction but to the best of our knowledge has not previously been used in the context of hashing or derandomization. This allows us to obtain running time that is essentially the same as our fast randomized algorithm when f is polynomial in n.

These ideas give the deterministic runtime in Theorem 1.2. Details are given in Section 5. Independently, Karthik and Parter [CP21] used similar ideas but in a more sophisticated manner to provide improved derandomizations for a number of related combinatorial objects, and their techniques when applied to our algorithm make it possible to remove the restriction that f is at least polynomial in n.

2 Preliminaries and Notation

We will use $\tilde{O}(\cdot)$ to suppress polylogarithmic (in n) factors. For any integer $a \geq 1$, let $[a] = \{1, 2, \dots, a\}$. Given an edge-weighted graph $G = (V, E, w)$, let $d_G(u, v)$ denote the shortest-path distance from u to v in G according to the weight function w and let $d_G^*(u, v)$ denote the unweighted distance (minimum number of hops) from u to v in G.

Many of our algorithms are randomized, and so we make claims that hold *with high probability*. Formally, this means that they hold with probability at least $1 - 1/n$.[1]

We will use the following Chernoff bounds (see [DP09]).

THEOREM 2.1. *Let $X = \sum_{i=1}^{n} X_i$, where X_i ($i \in [n]$) are independently distributed in $[0, 1]$. Then:*

- *For $0 < \epsilon < 1$: $\Pr[X < (1 - \epsilon)\mathbb{E}[X]] \leq \exp((-\epsilon^2/2)\mathbb{E}[X])$ and $\Pr[X > (1 + \epsilon)\mathbb{E}[X]] \leq \exp((-\epsilon^2/3)\mathbb{E}[X])$.*

- *If $t > 2e\,\mathbb{E}[X]$, then $\Pr[X > t] < 2^{-t}$.*

We will use the following structural lemma about fault-tolerant spanners, which was given explicitly in [DR20] but appeared implicitly in essentially all previous papers on fault-tolerant spanners. It essentially says that we only have to worry about spanning edges (not all pairs of nodes), and only edges for which the shortest path between the endpoints is the edge itself.

LEMMA 2.1. *Let $G = (V, E)$ be a graph with weight function w and let H be a subgraph of G. Then H is an f-VFT t-spanner of G if and only if $d_{H \setminus F}(u, v) \leq t \cdot w(u, v)$ for all $F \subseteq V$ with $|F| \leq f$ and $u, v \in V \setminus F$ such that $(u, v) \in E$.*

3 Optimal Fault-Tolerant Spanners in Polynomial Time

In this section we resolve the main open question left by [DR20, BP19, BDPW18] by proving Theorem 1.3: we give a polynomial time algorithm which constructs optimal-size vertex fault tolerant spanners. As discussed in Section 1.2.1, the algorithm itself is quite simple: we just use the greedy algorithm but test whether to include an edge by sampling subgraphs and checking whether the distance between the two endpoints is too large. This algorithm is given formally as Algorithm 1.

Algorithm 1 Basic f-VFT $(2k-1)$-Spanner Algorithm

Input: Graph $G = (V, E)$ on n nodes, edge weights $w : E \to \mathbb{R}^+$, integers $k \geq 1$ and $f \geq 1$.

1: $H \leftarrow (V, \emptyset)$
2: **for all** $e = (u, v) \in E$ in nondecreasing weight order **do**
3:    Sample $\alpha = c \log n$ subgraphs $\{\widehat{H_e^i} \subseteq H\}_{i \in [\alpha]}$, where each is an induced subgraph on a vertex set obtained by including u and v, and then each other node independently with probability $1/(2f)$.
4:    Let $\widehat{P_e}$ be the fraction of these subgraphs in which $d_{\widehat{H_e^i}}(u, v) > (2k - 1) \cdot w(u, v)$.
5:    **if** $\widehat{P_e} \geq 1/4$ **then**
6:       Add e to H
7: **return** H

The following definitions will be useful in our analysis. Let H be the final spanner, and let H' be an induced subgraph of H obtained by including each node, independently, with probability $1/(2f)$ (note that H' is only an analytical tool, not part of the algorithm). Let H_e (H'_e) denote the subgraph of H (H') containing only the edges considered strictly before e in the algorithm. For

---

[1]By changing the constants in the algorithm/analysis, all high probability claims we make can be made to hold with probability at least $1 - 1/n^c$ for any constant c. We choose $c = 1$ only for simplicity.

$e = (u, v)$, let

$$P_e := \Pr\left[d_{H'_e}(u, v) > (2k - 1) \cdot w(u, v) \mid u, v \in V(H')\right]$$

where the probability is over the random construction of H'. So $\widehat{P_e}$ in the algorithm is an experimental estimate of P_e, and we can bound its accuracy as follows:

LEMMA 3.1. *With high probability, for every edge $e \in E$, we have $\widehat{P_e} \in P_e \pm 1/8$.*

Proof. We will prove the lower bound $\widehat{P_e} \geq P_e - 1/8$; the upper bound is essentially identical. The random variable $\alpha \widehat{P_e}$ is the sum of α random variables $\left\{\widehat{P_e^i}\right\}$, where

$$\widehat{P_e^i} := \begin{cases} 0 & \text{if } d_{\widehat{H_e^i}}(u, v) \leq (2k - 1) \cdot w(u, v) \\ 1 & \text{if } d_{\widehat{H_e^i}}(u, v) > (2k - 1) \cdot w(u, v). \end{cases}$$

Thus we may apply Chernoff bounds (Theorem 2.1), giving:

$$\Pr\left[\alpha\widehat{P_e} < \alpha(P_e - 1/8)\right] = \Pr\left[\alpha\widehat{P_e} < \mathbb{E}\left[\alpha\left(\widehat{P_e} - 1/8\right)\right]\right]$$

$$\leq \Pr\left[\alpha\widehat{P_e} < \frac{7}{8}\,\mathbb{E}\left[\alpha\widehat{P_e}\right]\right]$$

$$\leq e^{-\frac{(1/8)^2}{2}\alpha\,\mathbb{E}\left[\widehat{P_e}\right]} \leq e^{-\alpha/128}$$

where the first equality follows by linearity of expectation, and the fact that by construction $P_e = \mathbb{E}\left[\widehat{P_e}\right]$. If we set $\alpha \geq 128 \cdot 3 \ln n$, then the probability is at most $1/n^3$. So, by a union bound over the $m \leq n^2$ edges in the input graph, the probability that the lower bound fails for *any* edge is at most $1/n$, proving the lemma. □

We are now ready to prove the properties of Algorithm 1.

LEMMA 3.2. *With high probability, Algorithm 1 returns an f-VFT $(2k - 1)$-spanner.*

Proof. Let $e = (u, v)$ be an edge considered by the algorithm, and suppose there exists a fault set $F \subseteq V \setminus \{u, v\}$ with $|F| \leq f$ such that

$$d_{H_e \setminus F}(u, v) > (2k - 1) \cdot w(e).$$

In the event that H'_e contains u, v but it does not contain any node in F, we thus also have

$$d_{H'_e}(u, v) > (2k - 1) \cdot w(e).$$

Thus P_e is *at least* the probability that none of the nodes in F survive in H'_e, which we may bound:

$$P_e \geq (1 - p)^{|F|} \geq \left(1 - \frac{1}{2f}\right)^f \geq 1/2.$$

By Lemma 3.1, with high probability we have $\widehat{P_e} \geq P_e - 1/8 \geq 1/2 - 1/8 > 1/4$, and so we add e to H in the algorithm. So for any edge e *not* added to the spanner, no such fault set F exists. It then follows from Lemma 2.1 that H is an f-VFT $(2k-1)$-spanner. □

LEMMA 3.3. *With high probability, $|E(H)| \leq O\left(f^{1-1/k}n^{1+1/k}\right)$.*

Proof. Recall that H' is an induced subgraph of H obtained by including every vertex independently with probability $1/(2f)$, and hence $\mathbb{E}[|V(H')|] = n/(2f)$. For each edge $(u, v) \in E(H')$, we say that (u, v) is *bad* if $d_{H'_e}(u, v) \leq (2k - 1) \cdot w(u, v)$, and otherwise (if $d_{H'_e}(u, v) > (2k - 1) \cdot w(u, v)$) we say that (u, v) is *good*. Let $H'' \subseteq H'$ be obtained by deleting all bad edges. We will now bound its expected number of edges $\mathbb{E}[|E(H'')|]$, conditioned on the high probability event from Lemma 3.1 holding, in two different ways:

- For any cycle C in H' with at most $2k$ edges, notice that the edge $(u, v) \in C$ considered last by the algorithm is bad, since there is a $u \rightsquigarrow v$ path around the cycle consisting of at most $2k - 1$ edges, each of weight at most $w(u, v)$. Thus (u, v) is removed in H''. It follows that H'' has *no* cycles with at most $2k$ edges; the folklore Moore Bounds then imply that

$$|E(H'')| = O\left(|V(H'')|^{1+1/k}\right).$$

Since each of the n nodes in H are included in H'' independently with probability $1/(2f)$, we have

$$(3.3) \qquad \mathbb{E}[|E(H'')|] = O\left(\mathbb{E}\left[|V(H'')|^{1+1/k}\right]\right)$$

$$= O\left(\mathbb{E}[|V(H'')|]^{1+1/k}\right)$$

$$= O\left(\left(\frac{n}{f}\right)^{1+1/k}\right).$$

In particular, the reason we can pull the exponent outside the expectation in the second step is due to the following computation:

$$\left|\mathbb{E}\left[|V(H'')|^{1+1/k}\right] - \mathbb{E}[|V(H'')|]^{1+1/k}\right|$$

$$\leq \left|\mathbb{E}\left[|V(H'')|^2\right] - \mathbb{E}[|V(H'')|]^2\right|$$

$$= Var[|V(H'')|] = O(n/f)$$

and hence the difference between these two terms may be hidden in the O.

- For an edge $e = (u, v) \in E(H)$, the probability that e survives in H'' may be decomposed as the probability that it survives in H', times the

probability that it survives in H'' given that it survives in H'. This gives:

$$
\begin{aligned}
&\Pr\left[(u,v) \in E(H'')\right] \\
&= \Pr\left[u, v \in V(H')\right] \\
&\qquad \times \Pr\left[(u,v) \text{ is good} \mid u, v \in E(H')\right] \\
&= \Theta(1/f^2) \cdot P_e.
\end{aligned}
$$

Since (u,v) was added to H, we have $\widehat{P_e} \geq 1/4$, and so (since we condition on Lemma 3.1) we have $P_e \geq 1/8$. Hence the probability of $(u,v) \in E(H'')$ is $\Theta(1/f^2)$. By linearity of expectations, we then have

$$
(3.4) \qquad |E(H'')| = \Theta\left(\frac{|E(H)|}{f^2}\right).
$$

Combining (3.3) and (3.4), we have

$$
\Omega\left(\frac{|E(H)|}{f^2}\right) = \mathbb{E}\left[|E(H'')|\right] = O\left(\left(\frac{n}{f}\right)^{1+1/k}\right)
$$

and so, comparing the left- and right-hand sides and rearranging, we get

$$
|E(H)| = O\left(n^{1+1/k} f^{1-1/k}\right)
$$

as desired. $\qquad\square$

LEMMA 3.4. *Algorithm 1 runs in polynomial time.*

Proof. We first need to sort the edges by weight, which takes at most $O(m \log n)$ time. Then for each edge $e \in E$, we must sample $\Theta(\log n)$ independent subgraphs and then run a single-source shortest path computation in each. Sampling such a subgraph takes $O(n + |E(H)|) \leq O(f^{1-1/k} n^{1+1/k})$ time (using the bound on $|E(H)|$ from Lemma 3.3). The running time of the shortest path computation on the subgraph is at most $O(|E(H)| \log n)$,[2] which is at most $O(f^{1-1/k} n^{1+1/k} \log n)$. Since we repeat $O(\log n)$ times per round, and we have m total rounds in the algorithm, the total runtime is $O(m f^{1-1/k} n^{1+1/k} \log^2 n)$. $\qquad\square$

4 An Even Faster Randomized Algorithm

Algorithm 2 is a bit more complicated than Algorithm 1, but it is significantly faster: in terms of runtime, it essentially turns the $O(m)$ factor in Lemma 3.4 into an $O(k^2 n)$ factor. In Algorithm 1, up to a logarithmic

---

[2]This estimate is conservative; except in a small range of parameters the subgraph is much smaller than H and thus the running time of the shortest path computation is dominated by the time needed to sample the subgraph in the first place.

factor the runtime per sampled subgraph is dominated by the time required to compute the subgraph rather than the time to measure distances on the subgraph, since sampling the subgraph requires time linear in $|E(H)|$ while distances are computed in the subgraph itself, which has fewer edges than H. This leads to essentially $O(|E(H)| \cdot m \log n)$ total runtime, since we sample $O(m \log n)$ subgraphs in total: $O(\log n)$ subgraphs in each round, and m total rounds. In our new algorithm, we improve this by (1) pre-sampling $\Theta(f^3 \log n)$ subgraphs and using them in each round, and (2) measuring relevant distances on these subgraphs using a certain incremental dynamic algorithm, rather than recomputing from scratch. See Section 1.2.2 for a more detailed overview. We now state the algorithm formally as Algorithm 2.

DEFINITION 4.1. *Let* $G = (V, E)$ *be an undirected graph, and let* $k \geq 1$ *be an integer. The* layered graph G^{2k} *is the directed graph with vertex set* $V \times [2k]$ *and edges*

$$
((u, i), (v, i+1)) \text{ for all } i \in [2k-1], (u, v) \in E, \text{ and}
$$
$$
((u, i), (u, i+1)) \text{ for all } i \in [2k-1], u \in V.
$$

Notice that the *unweighted* distance between u and v in G is at most $2k-1$ if and only if there is a path from $(u, 1)$ to $(v, 2k)$ in G^{2k}.

THEOREM 4.1. ([ITA86]) *There is a data structure which takes* $O(n^2)$ *time to initialize on an empty n-node graph, and which can then support directed edge insertions in* $O(n)$ *time (amortized) and reachability queries (answering "is there currently a* $u \rightsquigarrow v$ *path in the graph?") in* $O(1)$ *time.*

Like for Algorithm 1, we let H be the final spanner, and now we let H' be a uniform random subgraph *among those selected in the preprocessing phase*. Let H_e (H'_e) denote the subgraph of H (H') containing only the edges considered strictly before e in the algorithm. We note that we do not have separate analogous definitions of $\widehat{P_e}, P_e$ this time: the relevant probability P_e is computed *exactly* by the algorithm. We start our analysis with the following technical lemma:

LEMMA 4.1. *With high probability over the choice of random subgraphs in the preprocessing phase, for every* $e = (u, v) \in E$ *and* $F \subseteq V \setminus \{u, v\}$ *with* $|F| \leq f$, *we have:*

1. $|L_e| = O(f \log n)$

2. $|\{i \in L_e \mid F \cap V_i = \emptyset\}| = \Omega(f \log n)$

Algorithm 2 Faster f-VFT $(2k-1)$-Spanner Algorithm

Input: Graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}^+$, integers $k \geq 1$ and $f \geq 1$.
　　{**preprocessing phase**}
1: $H \leftarrow (V, \emptyset)$
2: **for** $i = 1$ **to** $\alpha = cf^3 \log n$ **do**
3: 　　Create V_i by including every vertex of V independently with probability $1/(2f)$
4: 　　Let $H_i = (V_i, \emptyset)$
5: 　　Create the layered graph H_i^{2k}, and initialize the data structure of Theorem 4.1 for H_i^{2k}
6: **for all** $e = (u, v) \in E$ **do** Let $L_e = \{i \mid u, v \in V_i\}$

　　{**main greedy algorithm**}
7: **for all** $e = (u, v) \in E$ in nondecreasing weight order **do**
8: 　　**for all** $H_i \in L_e$ **do**
9: 　　　　Query whether there is a $(u, 1) \rightsquigarrow (v, 2k)$ path in H_i^{2k}
10: 　　Let P_e be the fraction of subgraphs $H_i \in L_e$ where the query returns NO
11: 　　Let $0 < \tau < 1$ be an absolute constant that we choose in the analysis
12: 　　**if** $P_e \geq \tau$ **then**
13: 　　　　Add e to H
14: 　　　　**for all** $i \in L_e$ **do**
15: 　　　　　　Add e to H_i
16: 　　　　　　Update the connectivity data structure for H_i^{2k} by inserting the edges $((u, j), (v, j+1))$ and $((v, j), (u, j+1))$ in H_i^{2k} for all $j \in [2k-1]$
17: **return** H

Proof. For the first part, by linearity of expectations the expected number of sets V_i that contain both u and v is exactly $\alpha/(2f)^2 = (c/4)f \log n$. Applying Chernoff bounds (Theorem 2.1), we have (for sufficiently large c):

$$\Pr[|L_e| > ce \cdot f \log n] = \Pr[|L_e| > 4e\,\mathbb{E}[|L_e|]]$$
$$< 2^{-ce \cdot f \log n} < 1/(2n^3)$$

A union bound over the $m \leq n^2$ edges in the graph implies that $|L_e| = O(f \log n)$ for all e, simultaneously, with probability at least $1 - 1/(2n)$.

For the second part of the lemma, for any $e = (u, v) \in E$ and $F \subseteq V \setminus \{u, v\}$ with $|F| \leq f$, and for any $i \in [\alpha]$, we have

$$\Pr[i \in L_e \text{ and } F \cap V_i = \emptyset] = \Pr[u, v \in V_i] \cdot \Pr[F \cap V_i \neq \emptyset]$$

since the two probabilities on the right-hand side consider independent events (since $u, v \notin F$). Thus we may continue

$$\Pr[i \in L_e \text{ and } F \cap V_i = \emptyset] = \left(\frac{1}{4f^2}\right)\left(1 - \frac{1}{2f}\right)^{|F|}$$
$$\geq \frac{1}{4f^2}\left(1 - \frac{1}{2f}\right)^f$$
$$\geq \frac{1}{8f^2}.$$

By linearity of expectations,

$$\mathbb{E}[|\{i \in L_e \mid F \cap V_i = \emptyset\}|] \geq \frac{\alpha}{8f^2} = \left(\frac{c}{8}\right)f \log n.$$

Again by Chernoff bounds, we have

$$\Pr\left[|\{i \in L_e \mid F \cap V_i = \emptyset\}| < \left(\frac{1}{2}\right)\left(\frac{c}{8}\right)f \log n\right]$$
$$\leq \exp\left(\left(-\frac{1}{8}\right)\left(\frac{c}{8}\right)f \log n\right) \leq \frac{1}{2n^{4f}}$$

(where the last inequality is assuming sufficiently large c).

Taking a union bound over all $\leq n^f$ possible choices of F and all $m \leq n^2$ edges, we have

$$|\{i \in L_e \mid F \cap V_i = \emptyset\}| = \Omega(f \log n)$$

for all choices of e, F, simultaneously, with probability $\geq 1 - 1/(2n)$. Hence, by an intersection bound, the two parts of the lemma hold jointly with high probability. $\square$

We are now ready to prove the properties of Algorithm 2:

LEMMA 4.2. *With high probability, Algorithm 2 returns an f-VFT $(2k-1)$-spanner.*

Proof. Let $e = (u, v)$. As with our proof of correctness of Algorithm 1 (Lemma 3.2), by Lemma 2.1 we just need to show that when the algorithm considers e, if there is a fault set F for which

$$d_{H_e \setminus F}(u, v) > (2k-1) \cdot w(e),$$

then the algorithm adds e to H. Notice that this implies

$$d^*_{H_e \setminus F}(u, v) > 2k-1,$$

since by construction the weight of every edge in H_e is no larger than $w(e)$ (recall from Section 2 that d^* denotes the unweighted distance). By Lemma 4.1, with high probability a constant fraction of the $i \in L_e$ have $F \cap V_i = \emptyset$ and thus we have $d^*(u, v) > 2k-1$ in the corresponding subgraphs. Thus P_e is at least an absolute constant; by setting τ less than this constant, we will add e to H. $\square$

LEMMA 4.3. *With high probability, $|E(H)| \leq O\left(f^{1-1/k} n^{1+1/k}\right)$.*

Proof. This loosely follows the proof of Lemma 3.3. For each edge $(u, v) \in E(H')$, let us say:

$$(u, v) \text{ is } bad \text{ if } d^*_{H_{e'}}(u, v) \leq 2k-1$$
$$(u, v) \text{ is } good \text{ if } d^*_{H_{e'}}(u, v) > 2k-1.$$

Let $H'' \subseteq H'$ be obtained by deleting all bad edges. We now again bound $\mathbb{E}[|E(H'')|]$ in two ways:

- By essentially the same argument as in Lemma 3.3, H'' has no cycles on $\leq 2k$ edges, and thus

$$\mathbb{E}\left[|E(H'')|\right] = O\left(\mathbb{E}\left[|V(H'')|^{1+1/k}\right]\right)$$
$$= O\left(\mathbb{E}\left[|V(H'')|\right]^{1+1/k}\right)$$

where we may pull the exponent outside the expectation by the same argument as in Lemma 3.3. When we choose random subgraphs in the preprocessing phase, the *total* number of nodes added to all subgraphs can be viewed as the sum of independent binary random variables, and the expectation is $\Theta(nf^2 \log n)$. Thus, by Chernoff bounds, with high probability over the choice of random subgraphs in the preprocessing phase, we do indeed have $\Theta(nf^2 \log n)$ total nodes in our subgraphs. Conditioned on this high probability event, since we have exactly $\alpha = \Theta(f^3 \log n)$ subgraphs, the average sampled subgraph H_i has $|V(H_i)| = \Theta(n/f)$.

We then have $\mathbb{E}[|V(H'')|] = \Theta(n/f)$, and so with high probability

$$\mathbb{E}\left[|E(H'')|\right] = O\left(\left(\frac{n}{f}\right)^{1+1/k}\right)$$

(where the expectation is only over the choice of H' among the subgraphs sampled in the preprocessing phase).

- By Lemma 4.1, with high probability over the choice of random subgraphs, we have $|L_e| = \Theta(f \log n)$ for all edges e. In this event, whenever we add an edge $e = (u, v)$ to the spanner we also add the edge to $\Theta(f \log n)$ out of the $\alpha = \Theta(f^3 \log n)$ subgraphs. Moreover, by construction, in at least a constant τ fraction of these subgraphs H_i, our $(u, 1) \rightsquigarrow (v, 2k)$ path query returns NO. It follows that the current unweighted $u \rightsquigarrow v$ distance in these subgraphs is $> 2k-1$, and hence e is a *good* edge in $\Theta(\tau \cdot f \log n) = \Theta(f \log n)$ subgraphs. So the total number of good edges among the $\Theta(f^3 \log n)$ subgraphs is $\Theta(|E(H)|f \log n)$. We thus have

$$\mathbb{E}\left[|E(H'')|\right] = \Theta\left(\frac{|E(H)|}{f^2}\right).$$

Combining these, we have

$$\Omega\left(\frac{|E(H)|}{f^2}\right) = \mathbb{E}\left[|E(H'')|\right] = O\left(\left(\frac{n}{f}\right)^{1+1/k}\right),$$

and again the lemma follows by comparing the left- and right-hand sides and rearranging. $\square$

LEMMA 4.4. *The expected running time of Algorithm 2 is at most*

$$O\left(k^2 f^{1-1/k} n^{2+1/k} \log n + mf^2 \log n\right).$$

Proof. We first analyze the preprocessing phase. Note that without loss of generality, $m \geq f^{1-1/k} n^{1+1/k} \geq fn$ or else we are already finished (we can simply return the input graph).

We can create the V_i sets in time $O(nf^3 \log n) = O(mf^2 \log n)$ by flipping $\alpha = O(f^3 \log n)$ weighted coins for each vertex. Once we do this, we may assume that every vertex v has a *sorted* list L_v of the values of i for which $v \in V_i$. In expectation, each of the layered graphs H_i^{2k} has $O(kn/f)$ vertices, so we can then create the layered graphs in expected time $O((kn/f)f^3 \log n) = O(knf^2 \log n) = O(kf^{1-1/k} n^{2+1/k} \log n)$ (by linearity of expectations). Note that this includes the initial

edges of the form $((u, i), (u, i+1))$ in each of the layered graphs. Initializing the data structure of Theorem 4.1 for each of the layered graphs takes time $O((kn/f)^2)$ in expectation (since the number of nodes is binomial random variable and $\mathbb{E}[X^2] \leq O(\mathbb{E}[X]^2)$ for any binomial random variable X), and thus the total expected time to initialize these data structures is $O((kn/f)^2 f^3 \log n) = O(k^2 f n^2 \log n) \leq O(k^2 f^{1-1/k} n^{2+1/k} \log n)$. We also need to insert all of the initial edges of the form $((u, i), (u, i+1))$ into these data structures, which takes expected time $O(f^3 \log n \cdot (kn/f) \cdot (kn/f)) = O(k^2 f n^2 \log n) \leq O(k^2 f^{1-1/k} n^{2+1/k} \log n)$ (again using the square of a binomial random variable). Hence we can construct and initialize the vertex sets and the needed data structures in expected time at most

$$O(mf^2 \log n) + O(kf^{1-1/k} n^{2+1/k} \log n)$$
$$+ O(k^2 f^{1-1/k} n^{2+1/k} \log n)$$
$$+ O(k^2 f^{1-1/k} n^{2+1/k} \log n)$$
$$= O(mf^2 \log n) + O(k^2 f^{1-1/k} n^{2+1/k} \log n).$$

To create the L_e sets we need to be a little more careful, since doing it naively (looping through each edge and each $i \in [\alpha]$ and checking if both endpoints are in V_i) would take $O(mf^3 \log n)$ time. But we can speed this up: since every vertex v has L_v in a sorted list, for each edge $e = (u, v)$ we can just do a single pass through L_u and L_v to compute $L_e = L_u \cap L_v$. Thus this can be done in $O(|L_u| + |L_v|)$ time, and since $|L_v| \leq O(f^2 \log n)$ with high probability for every $v \in V$, this takes time $O(f^2 \log n)$ per edge and thus $O(mf^2 \log n)$ total.

Putting all of this together, we get that the preprocessing takes time $O(k^2 f^{1-1/k} n^{2+1/k} \log n + mf^2 \log n)$.

We now analyze the main loop. For every $e = (u, v) \in E$, the algorithm performs a connectivity query in $O(f \log n)$ of the layered subgraphs. By Theorem 4.1, this takes $O(mf \log n)$ total time. When we decide to add an edge e to the spanner (which happens at most $O(f^{1-1/k} n^{1+1/k})$ times by Lemma 4.3), we have to do $O(k)$ insertions into each of the $O(f \log n)$ layered graphs in L_e. The amortized cost of each insertion is $O(kn/f)$ by Theorem 4.1 (since the number of nodes in each layered graph is $O(kn/f)$ by Lemma 4.1), and hence the total time of all insertions is $O(f^{1-1/k} n^{1+1/k} (kn/f) kf \log n) = O(k^2 f^{1-1/k} n^{2+1/k} \log n)$. This is asymptotically larger than $m \log n$ since $m < n^2$, and hence the running time of the main loop is $O(k^2 f^{1-1/k} n^{2+1/k} \log n)$. □

5 Deterministic Algorithm

We now design a deterministic algorithm by derandomizing Algorithm 2. Recall that Algorithm 2 uses randomization in the preprocessing phase to create

$\Theta(f^3 \log n)$ vertex sets. We will derandomize this by *deterministically* creating sets with the same properties by using appropriately chosen hash functions. This results in a deterministic algorithm whose running time depends on the size of the hash family that we use. If we use universal or pairwise-independent hash functions, then we end up paying additional factors of n in the running time. We can improve this by using *almost*-universal hash functions, since our analysis is robust to changes in constants. Standard constructions then give the same dependence on n as in Algorithm 2, but polynomially worse dependence on f. But in the most important regime where f is polynomial in n, we can use ideas from *message authentication codes* in cryptography to design a hash family which is significantly more efficient, allowing us to get running time that is essentially *identical* to the randomized algorithm!

5.1 Set System Intuitively, we want sets which "act like" the $\Theta(f^3 \log n)$ random sets of Algorithm 2. So they should each have size about n/f, there shouldn't be too many sets in each L_e (each edge shouldn't be in too many of the subsets), and for every fault set F a constant fraction of the sets in L_e should not intersect F. We will proceed somewhat similarly to the approach of [Par19], who needed a set system where for all sets A of some size a and all sets B of some size b, there was at least one set in the system which contained all of A and none of B. This is similar to what we want, but differs in two important respects: we are only concerned with the special case of $a = 2$, but we want not just that there *exists* a set in the system which contains all of A and none of B, but that a *constant fraction* of the sets in the system which contain A do not contain any of B.

Trying to apply [Par19] as a black box, or even using the construction from [Par19], gives highly suboptimal bounds: the number of sets we would need would be exponential in f.[3] So we need to change the construction. And in order to optimize the running time we will hash onto a smaller range (approximately f, whereas using [Par19] would hash onto approximately f^2) and will use a different hash family (at least for the regime where f is polynomial in n).

5.1.1 Almost Universal Hashing Parter [Par19] began with "almost-pairwise independent" hash families. We will use a slightly different (but related) def-

---

[3]It is worth noting, though, that it is not hard to change the construction of [Par19] to give bounds that are polynomial in f: one simply needs to modify the construction in their Lemma 17 to define sets based on a hash value *being* some value, as opposed to the given construction which defines sets based on a hash value *not being* some value.

inition of "almost-universal" hash families, which is a weaker requirement but is sufficient for our needs and will allow us to design faster algorithms than if we required almost-pairwise independence.

DEFINITION 5.1. *A family $\mathcal{H} = \{h : U \to R\}$ is δ-almost universal if:*

1. *For all $x_1, x_2 \in U$ with $x_1 \neq x_2$: $\Pr_{h \sim H}[h(x_1) = h(x_2)] \leq \frac{\delta}{|R|}$, and*

2. *For all $y \in R$ and $h \in \mathcal{H}$, $|\{x \in U : h(x) = y\}| \leq O(|U|/|R|)$.*

3. *Each $h \in \mathcal{H}$ can be described with $\tilde{O}(1)$ bits and can be evaluated in $\tilde{O}(1)$ time.*

Setting $\delta = 1$ recovers the standard definition of universal hash families. The second and third parts of this definition are not always part of the standard definition of universality, but easily follow from most standard constructions (and, in particular, from the constructions that we will use). The third property implies that in time $\tilde{O}(|\mathcal{H}||U|)$ we can compute all hash functions from the family on all elements of the domain.

We will use two different constructions of δ-almost universal hash families, one which works for all regimes and one which only gives meaningful bounds when the size of the range is polynomial in the size of the domain. The first construction we will use is the following:

THEOREM 5.1. *For every $\delta > 1$, there is a δ-almost universal hash family $\mathcal{H}$ with $O(poly(1/(\delta - 1))|R|^4 \log^2 |U|)$ functions.*

To the best of our knowledge, this theorem does not appear explicitly anywhere in the literature, since most papers just bound the number of random bits as $O(\log |R| + \log \log |U|)$. Since the constant is unspecified, this is not enough to prove Theorem 5.1. But Theorem 5.1 can easily be derived from Theorem 2 of [AGHP92] by using $N = |U| \log |R|$ bits to define a hash function and setting $k = 2 \log |R|$ and $\epsilon = (\delta - 1)/|R|^2$.

If $|R|$ is close to $|U|$, we can use a different construction based on ideas from the cryptography literature, and in particular from *Message Authentication Codes* (MACs). This construction and analysis is essentially standard (see [KR09, dB93, Tay94, BJKS93]), but to the best of our knowledge has not been explicitly phrased as a hash function before in the literature. We will actually use a slightly weaker (and thus more efficient) version of the standard construction since we only need almost-universality, not almost-pairwise independence. We give the proof for completeness.

THEOREM 5.2. *There is a $\left\lceil \frac{\log |U|}{\log |R|} \right\rceil$-almost universal hash family $\mathcal{H}$ with $O(|R|)$ functions.*

Proof. Without loss of generality, let $U = \{0, 1\}^u$ and let $R = \{0, 1\}^r$ with u divisible by r, and we interpret U as $\mathbb{F}_{2^u}$ and R as $\mathbb{F}_{2^r}$. A single hash function is defined by a single element $a \in \mathbb{F}_{2^r}$. Given $x \in U$, we split x into u/r *chunks* $x_0, x_1, x_{(u/r)-1}$, each of which has r bits and so is an element of $\mathbb{F}_{2^r}$. This defines a polynomial $M_x(a) = \sum_{i=0}^{(u/r)-1} x_i a^i$ of degree $(u/r) - 1$. Given an element $a \in \mathbb{F}_{2^r}$, we define a hash function $h_a(x) = M_x(a)$.

Let $\mathcal{H} = \{h_a : a \in \mathbb{F}_{2^r}\}$. Clearly $|\mathcal{H}| = 2^r = |R|$. It is also not hard to see that $\mathcal{H}$ is a u/r-almost universal family. To see this, let $x, y \in U$ with $x \neq y$. Then $h_a(x) = h_a(y)$ if and only if $M_x(a) = M_y(a)$, which is equivalent to $M_x(a) - M_y(a) = 0$. Clearly $M_x(a) - M_y(a) = \sum_{i=0}^{(u/r)-1}(x_i - y_i)a^i$ is a non-zero polynomial (in a) of degree at most u/r, so there are at most u/r roots and thus the probability that we choose an a which satisfies this is at most $(u/r)/2^r = (u/r)/|R|$.

Clearly we can compute these functions quickly enough, so the third property of Definition 5.1 holds. The second property of Definition 5.1 also holds, since if we divide the possible $x \in U$ into equivalence classes by everything *except* their lowest-order chunk x_0 (so each class has 2^r elements and there are 2^{u-r} classes), then for every fixed a and every equivalence class there is exactly one element from the class which gets hashed to every possible value. $\square$

Since $\log |U| / \log |R|$ is a constant if R and U are polynomially related, Theorem 5.2 gives an $O(1)$-almost universal family in the case of $|R| \geq \text{poly}(|U|)$. We will only use this theorem in that regime.

5.1.2   Creating Our Sets We will use δ-almost universal hash families to create the subsets in the preprocessing stage of Algorithm 2 rather than creating these sets randomly. More formally, rather than create $O(f^3 \log n)$ sets independently, we will use the following construction.

Let $\mathcal{H}$ be a δ-almost universal hash family with domain V and range $[4\delta f]$.[4] We will create $|\mathcal{H}|\binom{4\delta f}{2} = \Theta(|\mathcal{H}|f^2)$ sets as follows: for every $h \in H$ and $y, z \in [4\delta f]$ with $y \neq z$, we let

$$V_{h,\{y,z\}} = \{v \in V : h(v) \in \{y, z\}\}$$

---

[4]This might seem strange in conjunction with Theorem 5.2, since then the range is a function of δ but δ is also a function of the range. We show how to set the parameters appropriately when we actually instantiate this hash family in the proof of Theorem 1.2.

In order to keep our previous notation, we will let $\alpha = |\mathcal{H}|\binom{4\delta f}{2}$ and will arbitrarily number these sets and refer to them as $V_1, \ldots, V_\alpha$. And as before, we will let $L_e = \{i \in [\alpha] : e \subseteq V_i\}$ be the sets which contain both endpoints of e. We first need the equivalent of Lemma 4.1, but now all bounds are deterministic.

LEMMA 5.1. *If $\mathcal{H}$ is δ-universal hash family for some constant $\delta > 1$ then there exist constants $c_1, c_2, c_3 > 0$ so that for sufficiently large n the sets $V_1, \ldots, V_\alpha$ have the following properties:*

1. *$|V_i| \leq c_1 n/f$ for all $i \in [\alpha]$,*

2. *$|L_e| \leq c_2 \delta |\mathcal{H}|$ for all $e \in E$, and*

3. *$|\{i \in L_e : F \cap V_i = \emptyset\}| \geq c_3 |\mathcal{H}|$ for all $e \in E$ and $F \subseteq V$ with $|F| \leq f$ and $F \cap e = \emptyset$.*

Proof. The second property of Definition 5.1 implies that each set has size at most $O(n/(4\delta f)) = O(n/f)$, so the first property is clearly true. For the second property, let $e = (u, v) \in E$, and let $h \in \mathcal{H}$. If $h(u) = h(v)$ then e is contained in $4\delta f - 1$ of the $\binom{4\delta f}{2}$ subsets defined by h, and if $h(u) \neq h(v)$ then e is contained in 1 of the subsets defined by h. By the first part of the definition of almost universal, $\Pr_{h \sim \mathcal{H}}[h(u) = h(v)] \leq \frac{\delta}{4\delta f} = \frac{1}{4f}$. Thus

$$|L_e| \leq \sum_{h \in \mathcal{H}} 1 + \sum_{h \in \mathcal{H}:h(u)=h(v)} (4\delta f - 1) \leq |\mathcal{H}| + \delta|\mathcal{H}|$$
$$= (1+\delta)|\mathcal{H}|.$$

For the third property, fix $e = (u, v) \in E$ and $F \subseteq V \setminus \{u, v\}$ with $|F| \leq f$. For every $x \in F$, let X_{xu} be the indicator random variable for the event that $h(x) = h(u)$, and similarly let X_{xv} be the indicator random variable for $h(x) = h(v)$. Note that by the definition of almost-universal, both of these random variables have expectation at most $\delta/(4\delta f) = 1/(4f)$. Let $Z \subseteq F$ be the set of elements of F that hash to the same value as u or v (so Z is a random subset). Then $|Z| \leq \sum_{x \in F} (X_{xu} + X_{xv})$, and thus

$$\mathbb{E}[|Z|] \leq \mathbb{E}\left[\sum_{x \in F}(X_{xu} + X_{xv})\right] = \sum_{x \in F}(\mathbb{E}[X_{xu}] + \mathbb{E}[X_{xv}])$$
$$\leq \sum_{x \in F}\frac{2}{4f} \leq \frac{1}{2}$$

So by Markov's inequality, the probability that $|Z| \geq 1$ is at most $1/2$. Thus in at least half of the hash functions from $\mathcal{H}$, nothing from F has the same hash value as u or v. If we have such an h, *and* if $h(u) \neq h(v)$, then this means that $F \cap V_{h,\{h(u),h(v)\}} = \emptyset$.

As discussed earlier, the number of hash functions in which $h(u) = h(v)$ is at most $|\mathcal{H}|/(4f)$. Thus the number of sets which contain both u and v but do not contain any element of F is at least

$$|\mathcal{H}|\left(\frac{1}{2} - \frac{1}{4f}\right) \cdot 1 \geq \frac{3}{8}|\mathcal{H}|$$

as claimed. $\square$

5.2 The Algorithm and Analysis
Our deterministic algorithm is the same as Algorithm 2, except:

1. Instead of sampling $O(f^3 \log n)$ sets, we use the above construction to create $O(|\mathcal{H}|f^2)$ sets which obey Lemma 5.1,

2. We set the threshold τ to c_3/c_2, i.e., we add e to the spanner if $P_e \geq c_3/c_2$ (where c_2 and c_3 are the constants from Lemma 5.1).

We now prove correctness and size bounds in essentially the same way as in the randomized algorithm. They just hold deterministically rather than with high probability.

LEMMA 5.2. *This algorithm returns an f-VFT $(2k-1)$-spanner.*

Proof. Let $e = (u, v)$. As with our proofs of correctness of Algorithms 1 and 2, by Lemma 2.1 we just need to show that if when the algorithm considers e there is a fault set F which is bad for e (i.e., $|F| \leq f$ and $d_{H \setminus F}(u, v) > (2k - 1)w(e)$), then the algorithm adds e to H.

Note that for any $i \in [\alpha]$ where $e \subseteq V_i$ but $F \cap V_i = \emptyset$, we have $d_{H_i}(u, v) > (2k - 1)w(e)$ (since removing F is enough to make the distance too large) and hence there is no $(2k - 1)$-hop path from u to v in H_i (since we consider the edges in nondecreasing weight order). This implies that our reachability query from $(u, 1)$ to $(v, 2k)$ in H_i^{2k} will return NO. Thus by Lemma 5.1, $P_e \geq \frac{c_3|\mathcal{H}|}{c_2|\mathcal{H}|} = \tau$, and so the algorithm will add e to the spanner. Lemma 2.1 then implies the theorem. $\square$

We can now prove our size bound.

LEMMA 5.3. *$|E(H)| \leq O(f^{1-1/k}n^{1+1/k})$.*

Proof. We use randomness in our analysis in the same way it was used in the proof of Lemma 4.2, but note that the algorithm itself is deterministic.

Choose an $i \in [\alpha]$ uniformly at random, and let $H' = H_i$. Then for each $(2k)$-cycle in H', remove the heaviest edge to get H''.

By part 1 of Lemma 5.1, we know that $|V(H'')| \leq O(n/f)$. Since H'' has no $(2k)$-cycles, this implies that

$$(5.5) \qquad |E(H'')| \leq O\left((n/f)^{1+1/k}\right).$$

Fix some $e = (u, v) \in E(H)$. Then the third property of Lemma 5.1 (with $F = \emptyset$) implies that $\Pr[e \in H'] \geq \Omega(|\mathcal{H}|/(|\mathcal{H}|f^2)) = \Omega(1/f^2)$. Conditioned on this, e fails to survive to H'' only if it was the heaviest edge on some $(2k)$-cycle in H'. Since the algorithm considers the edges in nondecreasing weight order, e is the heaviest edge on some $(2k)$-cycle if and only if when it was added by the algorithm the hop-distance between u and v in H' was at most $2k - 1$.

The probability of this happening (conditioned on e being in H') is by definition equal to $1 - P_e$, and so the probability that e *does* survive to H'' (conditioned on being in H') is precisely P_e. Since e was added by the algorithm we know that $P_e \geq c_3/c_2$, and thus the probability that e survives to H'' (conditioned on being in H') is at least c_3/c_2.

Hence

$$(5.6)$$
$$\mathbb{E}[|E(H'')|] \geq |E(H)| \cdot \Omega(1/f^2) \cdot (c_3/c_2) \geq \Omega(|E(H)|/f^2)$$

Combining (5.5) and (5.6) implies that

$$|E(H)| \leq O(f^2 \cdot (n/f)^{1+1/k}) = O(f^{1-1/k}n^{1+1/k})$$

as claimed. $\square$

We now analyze the running time. Clearly it depends on $|\mathcal{H}|$, but for now we will leave this as a parameter.

LEMMA 5.4. *The running time of the deterministic algorithm is at most*

$$\tilde{O}\left(f^{-1/k}n^{2+1/k}|\mathcal{H}| + m|\mathcal{H}|f\right).$$

Proof. We proceed as in Lemma 4.4 by first analyzing the preprocessing, and assuming without loss of generality that $m \geq f^{1-1/k}n^{1+1/k} \geq fn$.

Our set system has $O(|\mathcal{H}|f^2)$ sets. For each hash function in $\mathcal{H}$, we can create the f buckets in $\tilde{O}(n)$ time (by evaluating the hash function on each vertex in $\tilde{O}(1)$ time). Then we can build each of the $\Theta(f^2)$ sets for that function in time $O(n/f)$, so the time to create all of the $\Theta(|\mathcal{H}|f^2)$ sets is $\tilde{O}(|\mathcal{H}|fn)$. Creating the layered graphs takes additional $O(|\mathcal{H}|f^2(n/f)k) = O(|\mathcal{H}|nfk) = O(mk|\mathcal{H}|)$ time. Initializing all $O(|\mathcal{H}|f^2)$ data structures from Theorem 4.1 and inserting the initial edges takes time $O(|\mathcal{H}|f^2 \cdot (kn/f)^2) = O(k^2n^2|\mathcal{H}|) = \tilde{O}(f^{-1/k}n^{2+1/k}|\mathcal{H}|)$. As in the analysis of the fast randomized algorithm (Lemma 4.4), while creating the sets

we can record for each vertex a sorted list of which sets it is in, and then can create each L_e set by simple set intersection. Since each vertex is in $O(|\mathcal{H}|f)$ sets, this takes time $O(m|\mathcal{H}|f)$.

Thus our total preprocessing time is $\tilde{O}(f^{-1/k}n^{2+1/k}|\mathcal{H}| + m|\mathcal{H}|f)$.

We now analyze the main greedy loop. For every $e = (u, v) \in E$, the algorithm performs a connectivity query in $|L_e| \leq O(|\mathcal{H}|)$ of the layered subgraphs. By Theorem 4.1, this takes $O(m|\mathcal{H}|)$ total time. When we decide to add an edge e to the spanner (which happens at most $O(f^{1-1/k}n^{1+1/k})$ times by Lemma 5.3), we have to do $O(k)$ insertions into each of the $O(|L_e|) = O(|\mathcal{H}|)$ layered graphs in L_e. The amortized cost of each insertion is $O(kn/f)$ by Theorem 4.1 (since the number of nodes in each layered graph is $O(kn/f)$ by Lemma 5.1), and hence the total time of all insertions is $O(f^{1-1/k}n^{1+1/k}(kn/f)k|\mathcal{H}|) = O(k^2f^{-1/k}n^{2+1/k}|\mathcal{H}|)$. This is asymptotically larger than $m|\mathcal{H}|$ since $m < n^2$ and $f < n$, and hence the running time of the main loop is $O(k^2f^{-1/k}n^{2+1/k}|\mathcal{H}|)$. $\square$

This now finally allows us to prove Theorem 1.2, which we restate here for clarity.

THEOREM 1.2. *There is a deterministic algorithm which constructs an f-VFT $(2k - 1)$ spanner with at most $O(f^{1-1/k}n^{1+1/k})$ edges in time $\tilde{O}(f^{4-1/k}n^{2+1/k} + mf^5)$, and if $f = poly(n)$ (i.e., $f \geq n^c$ for some constant $c > 0$) then the running time improves to $\tilde{O}(f^{1-1/k}n^{2+1/k} + mf^2)$*

Proof. The fault tolerance and size bounds are from Lemmas 5.2 and 5.3. Using Lemma 5.4 with the almost universal construction of Theorem 5.1 (with $\delta = 2$) gives a running time of $\tilde{O}(k^2f^{-1/k}n^{2+1/k}f^4 + mf^5) = \tilde{O}(f^{4-1/k}n^{2+1/k} + mf^5)$.

On the other hand, if $f \geq n^c$ for some constant $c > 0$ then we want to use the almost universal construction of Theorem 5.2. But as discussed earlier, we have to be a little careful since if we use this construction then the range is a function of δ but δ is also a function of the range. So we need to show that we can set δ so that it is a constant and the range is $[4\delta f]$ (since that is the construction we used). We first set $\delta = \frac{\log n}{\log f} \leq 1/c$ and set the range of the hash family to be $4\delta f$. Then if we use Theorem 5.2 with this range, the theorem implies that this is a $\delta' = \frac{\log n}{\log(4\delta f)}$-almost universal family. But δ' is clearly at most δ, and hence it is also a δ-almost universal family. So we have, as required, a δ-almost universal family with range $[4\delta f]$ where $\delta \leq 1/c$ is a constant. When we use this family, Lemma 5.4 implies that the running time is $\tilde{O}(k^2f^{-1/k}n^{2+1/k}f + mf^2) = \tilde{O}(f^{1-1/k}n^{2+1/k} + mf^2)$. $\square$

6 Conclusion and Open Questions

In this paper we gave the first polynomial-time algorithm to construct optimal-size vertex fault-tolerant spanners. Our algorithm, after being optimized for running time, is also significantly faster than the previous best (non-optimal) polynomial time algorithm. We also derandomized our algorithm to get a deterministic algorithm, which is always polynomial time and, in the most interesting regime of f being polynomial in n, is just as fast (ignoring polylogarithmic factors) as our randomized algorithm.

There are still a number of tantalizing open problems involving fault tolerant spanners. Algorithmically, while we significantly optimized the running time to make it surprisingly efficient, even in the regime $f = O(1)$ it is still not as fast the fastest algorithms for non-FT spanners (e.g., [BS07] which has running time $\tilde{O}(m)$). These fast non-FT spanner algorithms are *not* the greedy algorithm, which seems to be unable to achieve such an efficient running time. Is it possible to compute optimal-size fault-tolerant spanners in time $\tilde{O}(m)$? Similarly, there has been significant work on computing spanners in other models of computation, most notably in distributed and parallel models (see [BS07, DGP07, DGP07, PY18, BDG$^+$20] for a small sampling of such results). The greedy algorithm is typically difficult to parallelize or implement efficiently distributedly (particularly in the presence of congestion), so there is the obvious question of computing optimal-size fault tolerant spanners efficiently in these models.

Next, we note that while all of our constructions and bounds work as well in the case of edge faults, they are not known to be optimal. The best-known lower bound on the size of an f-edge fault tolerant $(2k-1)$ spanner, proved in [BDPW18], is only $\Omega\left(f^{\frac{1}{2}(1-1/k)}n^{1+1/k}\right)$ rather than $\Omega\left(f^{1-1/k}n^{1+1/k}\right)$ as for vertex faults (for $k \geq 3$; for $k = 2$ the lower bounds are both $\Omega\left(f^{1/2}n^{3/2}\right)$, and hence the size bounds achieved here are optimal). Closing this gap for general k, by either giving improved upper bounds or improved lower bounds (or both), is probably the most important open question about fault-tolerant spanners.

Finally, there are some additional nice properties of the non-faulty greedy algorithm, and it would be interesting to determine whether these have desirable analogs for FT (slack-)greedy algorithms as well. In particular: the non-FT greedy algorithm gives optimal spanners for several important graph classes like Euclidean graphs and doubling metrics [LS19], it produces optimal spanners as measured by *lightness* [FS20], and there is experimental evidence that it performs particularly well on graphs encountered in practice [FG05].

References

[ABP91] Baruch Awerbuch, Alan Baratz, and David Peleg. Efficient broadcast and light-weight spanners. *Unpublished manuscript, November*, 1991.

[ADD$^+$93] Ingo Althöfer, Gautam Das, David P. Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993.

[AGHP92] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures & Algorithms*, 3(3):289–304, 1992.

[AP90] Baruch Awerbuch and David Peleg. Network synchronization with polylogarithmic overhead. In *Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on*, pages 514–522. IEEE, 1990.

[BBG$^+$20] Aaron Bernstein, Jan van den Brand, Maximilian Probst Gutenberg, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, and He Sun. Fully-dynamic graph sparsifiers against an adaptive adversary. *arXiv preprint arXiv:2004.08432*, 2020.

[BDG$^+$20] Amartya Shankha Biswas, Michal Dory, Mohsen Ghaffari, Slobodan Mitrović, and Yasamin Nazari. Massively parallel algorithms for distance approximation and spanners, 2020.

[BDPW18] Greg Bodwin, Michael Dinitz, Merav Parter, and Virginia Vassilevska Williams. Optimal vertex fault tolerant spanners (for fixed stretch). In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1884–1900. SIAM, 2018.

[BEH$^+$06] Georg Baier, Thomas Erlebach, Alexander Hall, Ekkehard Köhler, Heiko Schilling, and Martin Skutella. Length-bounded cuts and flows. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, pages 679–690, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[BJKS93] Jürgen Bierbrauer, Thomas Johansson, Gregory Kabatianskii, and Ben Smeets. On families of hash functions via geometric codes and concatenation. In *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '93, page 331–342, Berlin, Heidelberg, 1993. Springer-Verlag.

[BP19] Greg Bodwin and Shyamal Patel. A trivial yet optimal solution to vertex fault tolerant spanners. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, page 541–543,

New York, NY, USA, 2019. Association for Computing Machinery.

[BS07] Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Struct. Algorithms*, 30(4):532–563, 2007.

[CLPR10] Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. Fault tolerant spanners for general graphs. *SIAM J. Comput.*, 39(7):3403–3423, 2010.

[CP21] Karthik C.S. and Merav Parter. Deterministic replacement path covering. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, 2021.

[CZ04] Artur Czumaj and Hairong Zhao. Fault-tolerant geometric spanners. *Discrete & Computational Geometry*, 32(2):207–230, 2004.

[dB93] Bert den Boer. A simple and key-economical unconditional authentication scheme. *J. Comput. Secur.*, 2:65–72, 1993.

[DGP07] Bilel Derbel, Cyril Gavoille, and David Peleg. Deterministic distributed construction of linear stretch spanners in polylogarithmic time. In Andrzej Pelc, editor, *Distributed Computing*, pages 179–192, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[DK11] Michael Dinitz and Robert Krauthgamer. Fault-tolerant spanners: better and simpler. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*, pages 169–178, 2011.

[DP09] Devdatt Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, USA, 1st edition, 2009.

[DR20] Michael Dinitz and Caleb Robelle. Efficient and simple algorithms for fault-tolerant spanners. In *Proceedings of the 2020 ACM Symposium on Principles of Distributed Computing*, PODC '20, 2020.

[EEST08] Michael Elkin, Yuval Emek, Daniel A Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. *SIAM Journal on Computing*, 38(2):608–628, 2008.

[Erd64] Paul Erdős. Extremal problems in graph theory. In *In Theory of Graphs and its Applications, Proc. Sympos. Smolenice*, 1964.

[FG05] Mohammad Farshi and Joachim Gudmundsson. Experimental study of geometric t-spanners. In *European Symposium on Algorithms*, pages 556–567. Springer, 2005.

[FS20] Arnold Filtser and Shay Solomon. The greedy spanner is existentially optimal. *SIAM J. Comput.*, 49(2):429–447, 2020.

[Ita86] G.F. Italiano. Amortized efficiency of a path retrieval data structure. *Theoretical Computer Science*, 48:273 – 281, 1986.

[KP12] Michael Kapralov and Rina Panigrahy. Spectral sparsification via random spanners. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 393–398. ACM, 2012.

[KR09] Bhavana Kanukurthi and Leonid Reyzin. Key agreement from close secrets over unsecured channels. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, pages 206–223, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[LNS98a] Christos Levcopoulos, Giri Narasimhan, and Michiel Smid. Efficient algorithms for constructing fault-tolerant geometric spanners. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 186–195. ACM, 1998.

[LNS98b] Christos Levcopoulos, Giri Narasimhan, and Michiel Smid. Efficient algorithms for constructing fault-tolerant geometric spanners. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 186–195. ACM, 1998.

[LS19] Hung Le and Shay Solomon. Truly optimal euclidean spanners. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1078–1100. IEEE, 2019.

[Luk99] Tamas Lukovszki. New results on fault tolerant geometric spanners. *Algorithms and Data Structures*, pages 774–774, 1999.

[NS07] Giri Narasimhan and Michiel Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007.

[Par19] Merav Parter. Small Cuts and Connectivity Certificates: A Fault Tolerant Approach. In Jukka Suomela, editor, *33rd International Symposium on Distributed Computing (DISC 2019)*, volume 146 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:16, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Pel00] David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.

[PS89] David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.

[PU89a] David Peleg and Jeffrey D. Ullman. An optimal synchronizer for the hypercube. *SIAM J. Comput.*, 18(4):740–747, 1989.

[PU89b] David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *J. ACM*, 36(3):510–530, 1989.

[PY18] Merav Parter and Eylon Yogev. Congested clique algorithms for graph spanners. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, volume 121 of *LIPIcs*, pages 40:1–40:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[RZ11] Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2011.

[Tay94] Richard Taylor. An integrity check value algorithm for stream ciphers. In Douglas R. Stinson, editor, *Advances in Cryptology — CRYPTO' 93*, pages 40–48, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[TZ05] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1):1–24, 2005.