Improved Approximation for Fault-Tolerant 2-Spanner

Michael Dinitz
Johns Hopkins University

Yang Xiao
Johns Hopkins University

November 28, 2022

Abstract

The Sparsest 2-Spanner problem, where given an unweighted, undirected graph $G = (V, E)$ we try to find the smallest subgraph in which all pairwise distances are within a factor 2 of their original distances, has been known to admit an $O(\log(m/n))$-approximation since [12]. When we add *fault-tolerance* to this problem, it is known to admit an $O(\log n)$-approximation [9], which while independent of the amount of fault-tolerance can be much larger than $O(\log(m/n))$. It has been an open question whether the fault-tolerant 2-spanner problem admits an $O(\log(m/n))$-approximation. We partially answer this question: for k-fault tolerant 2-spanner, we give an algorithm which is an $O(\log(km/n))$-approximation, and under an additional assumption (that the minimum degree in G is at least k) is an $O(\log(m/n))$-approximation.

1 Introduction

Graph spanners are subgraphs that approximately preserve distances in graphs. More formally, they are defined as follows.

Definition 1. A subgraph G' of a graph $G = (V, E)$ is a *t-spanner* of G if $d_{G'}(u, v) \leq t \cdot d_G(u, v)$ for all $u, v \in V$.

The value t is known as the *stretch*. In this paper we will only be concerned with 2-spanners, so we will use "spanner" and "2-spanner" interchangeably.

Spanners were introduced by [13] and [14] in the context of distributed computing. While spanners have since found use throughout theoretical algorithms and in many applications (ranging from network routing [1, 16]to linear systems solving [11, 15], distributed computing still forms an important setting for spanners.

However, one important property that is not captured by this definition is the possibility of *failure*. Not only does fault-tolerance introduce interesting mathematical questions, due to their close connection to distributed computing and the prevalence of faults in distributed contexts it is particularly important for spanners to be tolerant to failures. The basic definition, introduced by Chechik et al. [7], is the following.

Definition 2. A subgraph G' of a graph $G = (V, E)$ is k-vertex fault tolerant (k-VFT) 2-spanner of G if

$$d_{G' \setminus F}(u, v) \leq 2 \cdot d_{G \setminus F}(u, v)$$

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

We note that we could have used edge faults rather than vertex faults, i.e., let $F \subseteq E$ with $|F| \leq k$. These behave differently for stretch values larger than 2 (see [5] for a discussion of the major differences), but for stretch 2 are equivalent. For completeness, we prove this equivalence in Appendix A. So for the remainder of the paper, we use "vertex fault-tolerant" and "fault-tolerant" interchangeable.

Since their introduction in [7], fault-tolerant spanners and related objects have been studied extensively [2–6, 9, 10]. However, the vast majority of this work has has focused on *existential* questions, particularly around sparsity, where we attempt to understand the best tradeoff between stretch and size that holds in all graphs. But for stretch values below 3, it is not possible to guarantee the existence of spanners with subquadratic edges, even without fault-tolerance, and so no tradeoff is possible.

This impossibility motivated Kortsarz and Peleg [12] to study the Minimum 2-Spanner problem, where instead of trying to find a universal tradeoff we instead attempt per-instance optimization. That is, given a graph $G = (V, E)$, the goal is to compute the smallest 2-spanner of G (measured by number of edges). While this is NP-hard, they showed that it could be approximated in polynomial-time to within an $O(\log(m/n))$-factor, where $m = |E|$ and $n = |V|$. In the fault-tolerant context, the related problem of Minimum k-Fault Tolerant 2-Spanner (given a graph, find the smallest possible k-fault tolerant 2-spanner) was introduced by [8], who gave an $O(k \log n)$-approximation, which was then improved to an $O(\log n)$-approximation (independent of k) by [9].

This leads to an obvious question: is there an $O(\log(m/n))$-approximation for Minimum k-Fault Tolerant 2-Spanner? Or is $O(\log n)$ the best we can hope for?

1.1 Our Results

In this paper we make the first progress on the k-fault tolerant 2-spanner problem since [9]: we give an $O(\log(km/n))$-approximation, and under a relatively mild assumption, that every node has degree $\Omega(k)$ in G (or even that the optimum k-fault tolerant 2-spanner has size $\Omega(kn)$), we improve this to an $O(\log(m/n))$-approximation. Unlike the previous algorithms for this problem [8,9], our algorithm is not based on rounding an LP relaxation, but is purely combinatorial: we show how to modify the greedy algorithm of [12] to take fault-tolerance into account. We also show that our analysis is tight by providing construction of a family of graphs for which our algorithm gives exactly $O(\log \frac{km}{n})$-approximation which is based on the construction described in [12].

2 Preliminaries

We begin with some basic notation. Given a graph $G = (V, E)$, we will typically use n to denote $|V|$ and m to denote $|E|$. For $U \subseteq V$, we let $E(U) = \{(v_1, v_2) \in E : v_1, v_2 \in U\}$ denote edges with both endpoints in U. We let $N_G(v) = \{u : (u, v) \in E\}$ denote the neighbors of v in G.

Let $E' \subseteq E$ be a set of edges. We say v *covers* $e = (u_1, u_2)$ using edges in E' if $(v, u_1) \in E'$ and $(v, u_2) \in E'$. We say that $e = (u_1, u_2)$ is covered α times by E' if there are at least α nodes that cover e using edges in E'.

The following alternative characterization of fault-tolerant 2-spanners was proved in [9], and will be particularly useful for us.

Theorem 1 ([9]). *Let $G = (V, E)$, and let $G' = (V, E')$ be a subgraph of G. Then G' is a k-VFT 2-spanner of G if and only if e is covered $k + 1$ times by E' for every $e \in E \setminus E'$.*

3 Algorithm

As mentioned in Section 1.1, our algorithm is based on modifying the greedy algorithm of [12] to the fault-tolerant setting. In particular, one can (at least informally) think of the algorithm of [12] as attempting to find the "cheapest" way of covering all edges of E at least once (or including them). Thanks to Theorem 1, we know that our goal is to essentially cover every edge at least $k + 1$ times.

Before formally giving our algorithm, we first set up some more notation. For $S, P \subset E$ and $U \subset V$, denote by $R(P, U)$ the set of edges in G induced by U restricted to P, namely, $R(P, U) = E(U) \cap P$.

Let h be a function defined from E to 2^V, let $P \subset E$, denote

$$cov_G(P, h, v) = |\{e \in R(P, N_G(v)) : v \notin h(e)\}|$$

to be the number of edges in P that is covered by v using edges in G with $v \notin h(e)$.

Define maximum density function to be:

$$\rho_G(P,S,h,v) = \max_{U \subseteq N_G(v)} \left\{ \frac{|\{e \in R(P,U) : v \notin h(e)\}| + \sum_{w \in U:(v,w)\notin S}(k+1-|h(v,w)|)}{|U|} \right\}$$

We can now formally give our algorithm, in Algorithm 1. After defining the algorithm formally, we provide some intuition.

Algorithm 1 Greedy Algorithm

```
Input: Given a Graph $G = (V,E)$ and Fault Tolerant Parameter $k$
  Set $H^U \leftarrow E; H^C \leftarrow \emptyset; H^S \leftarrow \emptyset; C^{cur} \leftarrow \{\emptyset : \forall e \in E\}$;
  while $\exists\ v \in V\ s.t.\ \rho_G(H^U, H^S, C^{cur}, v) \geq 1$ do
    choose $v = \operatorname{argmax}_{u \in V} \rho_G(H^U, H^S, C^{cur}, u)$;
    (Let the corresponding densest subset be $U_v$)
    $H^S \leftarrow H^S \cup \{(v,u) : u \in U_v\}$;
    for $e \in R(H^U, U_v)$ and $v \notin C^{cur}(e)$ do
      $C^{cur}(e) \leftarrow C^{cur}(e) \cup \{v\}$
      if $|C^{cur}(e)| == k+1$ then
        $H^C \leftarrow H^C \cup \{e\}$;
      end if
    end for
    $H^C \leftarrow H^C \cup H^S$;
    $H^U \leftarrow H^U \setminus H^C$;
  end while
  return $H^S \cup H^U$
```

We now provide some intuition. Throughout the algorithm, we maintain three sets of edges: H^U, H^C and H^S. The set H^S contains *spanner edges*, which means all the edges that were already added to the constructed spanner. The set H^C consists of all the edges that "have been taken care of" (which we also refer to as "covered edges"), which means each edge in H^C is either included H^S or is covered at least $k+1$ times by H^S. The set H^U contains all the edges that "have not been taken care of" (the "uncovered edges"): every edge in H^U is neither in the spanner, nor been covered at least $k+1$ times. We maintain a function C^{cur} to record the set of vertices that covers each edge using H^S.

Our algorithm operates by repeatedly performing the following operation. For every vertex v, we consider the edges in $R(H^U, N_G(v))$ (the set of uncovered edges induced by neighbors of v) that have not already been covered by v using H^S. In this graph we look for a subset U_v of maximum density. Then we choose the densest set among all the sets $\{U_v : v \in V\}$. If we choose U_w, then we add a star to H^S consisting of the edges between w and U_w. In this way, we cover all the edges in $R(H^U, U_w)$ (that have not been covered by v before) one additional time using v, while adding only a small number of edges ($|U_w|$) to the spanner. It is worth mentioning that we have to maintain the C^{cur} function since the same vertex might be repeatedly chosen, and an edge can only be covered one time by the same vertex.

Intuitively, the edge that can be covered one time by choosing w are given "value" of 1, the edge we add to H^S (say e) can be seen as being directly covered to $k+1$ times and thus are given $k+1-C^{cur}(e)$ credits, so the total value we are getting toward covering edges by U_w is $|e \in R(H^U, U_w) : w \notin C^{cur}(e)| + \sum_{x \in U_w:(x,w)\notin H^S}(k+1-|C^{cur}(x,w)|)$ (Note that each edge can only be added to the spanner for one time, therefore we shouldn't give anymore credit after it's chosen). Since we are trying to choose the most cost-effective subset in neighbors of w, the density function is defined as above. Note that this density function is a generalization of [12] since if $k=0$, then the density function described above is exactly equivalent to the one in [12] plus a constant.

4 Analysis

Let $r = (k+1)\frac{|E|}{|V|}$ and let $f = \log r$. Denote ρ_G^i to be the maximum density achieved in iteration i, then since H^U is non-increasing, $C^{cur}(e)$ is non-decreasing for every $e \in E$ in size at every step, we can have $\rho_G^i \geq \rho_G^j$ for $j > i$. We divide the iterations of the algorithm into f phases based on this density. The first phase starts in the first iteration, and consists of all iterations where $\rho_G(H^U, H^S, C^{cur}, v) \geq \frac{r}{2}$ is satisfied when we select v. Then for $i \geq 2$, phase i consists of iterations where $\frac{r}{2^i} \leq \rho_G(H^U, H^S, C^{cur}, v) < \frac{r}{2^{i-1}}$ is satisfied when we select v.

For the rest of the analysis, we will use the following notation:

1. V_i is the set of vertices that are chosen in phase i
2. H_i^S is the set of edges added to H^S in phase i, let $S_i = \cup_{i=1}^{i} H_i^S$
3. H_i^U is the set of edges left in H^U at the end of phase i
4. C_i as the function that records the set of vertices which are covering edge e using $\cup_{j\in[i]} H_j^S$ (edges added to spanner in the first i phases) for each $e \in E$ at the end of phase i.
5. $h_i^S(v)$ as the set of edges that are added (for the first time) to H^S when vertex v is chosen in phase i but did not exist in H^S before v is chosen in phase i.
6. $h_i^T(v)$ as the set of edges that are covered by v using $h_i^S(v)$ in phase i
7. $\sigma_i = \sum_{v\in V_i}\{|h_i^T(v)| + \sum_{e\in h_i^S(v)}(k+1-|C_i(e)|)\}$ as the total times of covers in phase i
8. Let $E' \subset E$. An optimal k-VFT 2-spanner for E' is a minimum subset $E'' \subset E$ such that every edge in $E' \setminus E''$ is covered by at least $k+1$ times. Let H^* be the optimal spanner for G.

Note that in our algorithm, we pick the vertex that gives the maximum density in each iteration, which intuitively reflects a strategy to cover the most edges while cost as less as possible to construct spanner. To quantify how much advantage this greedy strategy takes, we will first compare the number of covers we make with the number of edges we add to construct spanner in each iteration:

Lemma 2. *For every $1 \leq i \leq f$,*

$$\frac{\sigma_i}{|H_i^S|} \geq \frac{r}{2^i}$$

Proof. According to definition, in phase i, whenever we choose a vertex v, density $\rho(H^U, H^S, C^{cur}, v) \geq \frac{r}{2^i}$, therefore, for $v \in V_i$, we have:

$$\begin{aligned}
\frac{\sigma_i}{|H_i^S|} &= \frac{\sum_{v\in V_i}\{|h_i^T(v)| + \sum_{e\in h_i^S(v)}(k+1-|C_i(e)|)\}}{|H_i^S|} \\
&\geq \frac{\sum_{v\in V_i}\{|h_i^T(v)| + \sum_{e\in h_i^S(v)}(k+1-|C_i(e)|)\}}{\sum_{v\in V_i}|h_i^S(v)|} \\
&\geq \min_{v\in V_i}\left\{\frac{|h_i^T(v)| + \sum_{e\in h_i^S(v)}(k+1-|C_i(e)|)}{|h_i^S(v)|}\right\} \\
&\geq \frac{r}{2^i}
\end{aligned}$$

. □

Let $H_i^* \subset E$ be the optimal spanner for H_i^U, where each edge e in $H_i^U \setminus H_i^*$ is covered at least $k+1$ times. Denote $G_i^* = (V, H_i^*)$. It's easy to see that $|H^*| \geq |H_i^*|$.

Define

$$\begin{aligned}\rho_i^*(v) &= \frac{cov_{G_i^*}(H_i^U, C_i, v) + \sum_{w \in N_{G_i^*}(v):(v,w)\notin S_i}(k+1-C_i(v,w))}{|N_{G_i^*}(v)|} \\ &= \frac{\big|\{e \in R(H_i^U, N_{G_i^*}(v)) : v \notin C_i(e)\}\big| + \sum_{w \in N_{G_i^*}(v):(v,w)\notin S_i}(k+1-C_i(v,w))}{|N_{G_i^*}(v)|}\end{aligned}$$

Lemma 3. *For every* $1 \leq i \leq f$,

$$\rho_i^*(v) < \frac{r}{2^i}$$

Proof. Note that according to our definition of density,

$$\rho_{G_i^*}(H_i^U, S_i, C_i, v) = \max_{U \subset N_{G_i^*}(v)}\{\frac{\big|\{e \in R(H_i^U, U) : v \notin C_i(e)\}\big| + \sum_{w \in U:(v,w)\notin S_i}(k+1-C_i(v,w))}{|U|}\}$$

Therefore,

$$\rho_i^*(v) \leq \rho_{G_i^*}(H_i^U, H_i^S, C_i, v)$$

Meanwhile,

$$\begin{aligned}\rho_{G_i^*}(H_i^U, H_i^S, C_i, v) &= \max_{U \subset N_{G_i^*}(v)}\{\frac{\big|\{e \in R(H_i^U, U) : v \notin C_i(e)\}\big| + \sum_{w \in U:(v,w)\notin S_i}(k+1-C_i(v,w))}{|U|}\} \\ &\leq \max_{U \subset N_G(v)}\{\frac{\big|\{e \in R(H_i^U, U) : v \notin C_i(e)\}\big| + \sum_{w \in U:(v,w)\notin S_i}(k+1-C_i(v,w))}{|U|}\} \\ &= \rho_G(H_i^U, H_i^S, C_i, v) \\ &< \frac{r}{2^i}\end{aligned}$$

□

We now prove the following claim:

Lemma 4. *For every* $1 \leq i \leq f$,

$$\frac{\sum_{e \in H_i^U}(k+1-|C_i(e)|)}{|H^*|} < \frac{r}{2^{i-1}}$$

Proof. According to the definition of k-VFT 2-spanner, we can have:

$$\begin{aligned}\sum_{e \in H_i^U}(k+1-|C_i(e)|) &\leq \sum_{v \in V} cov_{G_i^*}(H_i^U, C_i, v) + \sum_{e \in H_i^* \setminus S_i}(k+1-|C_i(e)|) \\ &= \sum_{v \in V} cov_{G_i^*}(H_i^U, C_i, v) + \frac{1}{2}\sum_{v \in V}\sum_{u \in N_{G_i^*}(v):(u,v)\notin S_i}(k+1-|C_i((u,v))|) \\ &\leq \sum_{v \in V}\left\{cov_{G_i^*}(H_i^U, C_i, v) + \sum_{u \in N_{G_i^*}(v):(u,v)\notin S_i}(k+1-|C_i((u,v))|)\right\}\end{aligned}$$

Therefore,

$$\begin{aligned}
\frac{\sum_{e\in H_i^U}(k+1-|C_i(e)|)}{|H^*|} &\le \frac{\sum_{e\in H_i^U}(k+1-|C_i(e)|)}{|H_i^*|} \\
&\le \frac{\sum_{v\in V}\left\{cov_{G_i^*}(H_i^U, C_i, v) + \sum_{u\in N_{G_i^*}(v):(u,v)\notin S_i}(k+1-|C_i((u,v))|)\right\}}{|H_i^*|} \\
&= \frac{\sum_{v\in V}\left\{cov_{G_i^*}(H_i^U, C_i, v) + \sum_{u\in N_{G_i^*}(v):(u,v)\notin S_i}(k+1-|C_i((u,v))|)\right\}}{\frac{1}{2}\sum_{v\in V} N_{G_i^*}(v)} \\
&\le 2\cdot\max_{v\in V}\frac{cov_{G_i^*}(H_i^U, C_i, v) + \sum_{u\in N_{G_i^*}(v):(u,v)\notin S_i}(k+1-|C_i((u,v))|)}{N_{G_i^*}(v)} \\
&= 2\cdot\max_{v\in V}\rho_i^*(v)
\end{aligned}$$

Thus, by lemma 3, we have

$$\frac{\sum_{e\in H_i^U}(k+1-|C_i(e)|)}{|H^*|} < \frac{r}{2^{i-1}}$$

□

Corollary 5. $\frac{|H^U|}{|H^*|} < 2$

Proof.

$$\begin{aligned}
|H_i^U| = \sum_{e\in H_i^U} 1 &\le \sum_{e\in H_i^U}(k+1-|C_i(e)|) \\
\Rightarrow \frac{|H_i^U|}{|H^*|} &< \frac{r}{2^{i-1}} \\
\Rightarrow \frac{|H^U|}{|H^*|} &< 2
\end{aligned}$$

□

Lemma 6. *For every* $1 \le i \le f$,

$$\frac{|H_i^S|}{|H^*|} \le 2$$

Proof. By combining previous lemmas,

For $i > 1$, we have:

$$\frac{|H_i^S|}{|H^*|} \le \frac{\frac{2^i}{r}\cdot\sigma_i}{|H^*|} \le \frac{\frac{2^i}{r}\cdot\sum_{e\in H_{i-1}^U}(k+1-|C_i(e)|)}{|H_i^*|} < \frac{2^i}{r}\cdot\frac{r}{2^{i-1}} = 2$$

For $i = 1$, we have:

$$\frac{|H_1^S|}{|H^*|} \le \frac{\frac{2}{r}\cdot\sigma_1}{|H^*|} \le \frac{\frac{2}{(k+1)\frac{|E|}{|V|}}\cdot(k+1)|E|}{|V|} = 2$$

□

Corollary 7. $\frac{|H^S|}{|H^*|} = O(\log r)$

Proof. By lemma 6,

$$\frac{|H^S|}{|H^*|} = \frac{\sum_{i=1}^{f} |H_i^S|}{|H^*|} = 2\log r = O(\log r)$$

□

From Corollaries 5 and 7, we conclude our main result.

Theorem 8. *Algorithm 1 is an $O(log(\frac{km}{n}))$-approximation algorithm for k-vertex fault tolerant 2-spanner problem.*

Now, notice that in above analysis, we only expanded r in lemma 6. So if we re-define $r = \frac{|E|}{|V|}$, all above lemmas and corollaries hold except for lemma 6 and corollary 7.

Lemma 9. *Assume G has minimum degree at least $k+1$, let $r = \frac{|E|}{|V|}$, then for every $1 \le i \le f$,*

$$\frac{|H_i^S|}{|H^*|} \le 2$$

Proof. For $i > 1$, we have:

$$\frac{|H_i^S|}{|H^*|} \le \frac{\frac{2^i}{r} \cdot \sigma_i}{|H^*|} \le \frac{\frac{2^i}{r} \cdot \sum_{e \in H_{i-1}^U} (k+1-|C_i(e)|)}{|H_i^*|} < \frac{2^i}{r} \cdot \frac{r}{2^{i-1}} = 2$$

For $i = 1$, we have:

$$\frac{|H_1^S|}{|H^*|} \le \frac{\frac{2}{r} \cdot \sigma_1}{|H^*|} \le \frac{\frac{2}{\frac{|E|}{|V|}} \cdot (k+1)|E|}{(k+1)|V|} = 2$$

□

Then we immediately have:

Corollary 10. *Assume G has minimum degree at least $k+1$, let $r = \frac{|E|}{|V|}$, then $\frac{|H^S|}{|H^*|} = O(\log r)$*

Combining Corollaries 5 and 10 gives us another result.

Theorem 11. *Assume the minimum degree of input graph is at least $k+1$, Algorithm 1 is an $O(log(\frac{m}{n}))$-approximation algorithm for k-vertex fault tolerant 2-spanner problem.*

5 A Lower Bound

In this section, we provide construction of a family of graphs for which our algorithm gives exactly $O(log\frac{km}{n})$-approximation.

5.1 A Sketch of Construction

To get started, we first discuss one possible condition that our algorithm produces a huge gap. Denote vertices with degree less than $k+1$ by V_{low}, vertices with degree at least $k+1$ by V_{high}, edges with at least one endpoint in V_{low} by E_{low}, edges with both endpoints in V_{high} by E_{high}. Let $G_{low} = (V_{low}, E_{low})$ be low-degree subgraph, $G_{high} = (V_{high}, E_{high})$ be high-degree subgraph. Denote $|V_{low}| = n_l$, $V_{high} = n_h$, $|E_{low}| = m_l$, $|E_{high}| = m_h$. Note that, by 1, all edges in E_{low} should be included in any valid k-VFT 2-spanner.

Let OPT_{low} be the optimal spanner for G_{low}, ALG_{low} be the spanner generated by our algorithm for G_{low}, OPT_{high} and ALG_{high} are defined similarly. By 11, it is guaranteed that $\frac{|ALG_{high}|}{|OPT_{high}|} \le \log \frac{m_h}{n_h}$.

Now, if $m_l = O(n_l)$, $n_h = O(\sqrt{n_l \log n_l})$, $m_h = O(n_l \log n_l)$, k be $poly(\sqrt{n_l \log n_l})$, then, $\log \frac{m}{n} = O(\log \log n_l)$, however $\log k = O(\log n_l)$

Assuming our algorithm basically takes every edge in G_{high}, *i.e.* $|ALG_{high}| = O(n_l \log n_l)$, and assuming the gap is exactly $O(log \frac{km}{n}) = O(\log n_l)$, *i.e.* $|OPT_{high}| = O(\frac{n_l \log n_l}{\log \sqrt{n_l \log n_l}}) = O(n_l)$, then, $|ALG| = |ALG_{high}| + |ALG_{low}| = O(n_l \log n_l) + O(n_l) = O(n_l \log n_l)$, $|OPT| = |OPT_{high}| + |OPT_{low}| = O(n_l)$, our algorithm indeed gives a $O(\log n_l) = O(\log k) = O(\log \frac{km}{n})$ approximation instead of $O(\log \frac{m}{n})$, and the gap location is at the top.

5.2 Construction of Tight Example for $k = \sqrt{n/\log n}$

5.2.1 The Graph G_k

Based on the construction provided by [12], we give a slightly modified graph G_k for which our algorithm gives $O(\log k)$ approximation.

Let $q = 2^p$ for an integer p. Denote $q' = q - 4$, let $U = \{u_1, u_2, ..., u_{q'}\}$ and $W = \{w_1, w_2, ..., w_{q'}\}$. Break the set U into $p - 2$ subsets by successive halving, letting U_1 contain the first $\frac{q}{2}$ vertices, U_2 contain the next $\frac{q}{4}$ vertices, and so on. Also define two additional sets A and B, let $A = \cup_{i=1}^{k+1} A_i$ where $A_i = \{a_i(1), a_i(2), a_i(3), a_i(4)\}$, let $B = \cup_{i=1}^{k+1} B_i$ where $B_i = \{b_i(1), b_i(2), ..., b_i(p-2)\}$. The vertex set is $U \cup W \cup A \cup B$, note that the number of vertices is $O(q)$ as long as $k \leq \frac{q}{p}$. We shall further break each set U_i into four equal-sized subsets $U_i(j)$, for $j = 1, 2, 3, 4$. We now specify the edges:

1. (E_1) For $1 \leq i \leq k+1$, For $1 \leq j \leq 4$, $a_i(j)$ is connected to $W \cup \cup_x U_x(j)$
2. (E_2) For $1 \leq i \leq k+1$, For $1 \leq j \leq p-2$, $b_i(j)$ is connected to $W \cup U_j$
3. (E_3) The sets W and U are connected by a complete bipartite graph(that is, W and U are independent sets, and every vertex of W is connected with every vertex of U
4. (E_4) A and B are connected by a complete bipartite graph

5.2.2 The Approximation Ratio of the Greedy Algorithm on G_k

We shall now consider of the k-VFT 2-spanner problem on graph G_k defined above. First, we can observe that this graph has a k-VFT 2-spanner of size O(kq) by taking the edge subsets E_1 and E_4. Therefore we have that the optimal solution for G_k is O(kq).

However, we claim our algorithm will greedily take E_2 and E_4, which contains $O(kq \log q)$ edges. Our algorithm will successively select $b_1(1), b_2(1), ..., b_{k+1}(1)$, then $b_1(2), b_2(2), ..., b_{k+1}(2)$,...,and at last $b_1(p-2), b_2(p-2), ..., b_{k+1}(p-2)$ and add all the edges incidents on these vertices.

Claim 12. *The first k vertices to be selected by our algorithm is $b_1(1), b_2(1), ..., b_{k+1}(1)$*

Proof. Let us first compute bounds of density for every v in the initial situation. Consider a vertex $v \in A \cup B$, and denote its corresponding densest subgraph by U_v, then the density $\rho(H^U, \emptyset, C, v)$ can be represented by:

$$\begin{aligned}\rho(H^U, \emptyset, C, v) &= \frac{|E(U_v)| + (k+1)|U_v|}{|U_v|} \\ &= \frac{|E(U_v)|}{|U_v|} + k + 1\end{aligned}$$

which is exactly the density described in [12] plus a constant term. Since for any $1 \leq j \leq 4$, $a_1(j), a_2(j)..., a_{k+1}(j)$ shares the same neighbors, therefore

$$\rho(H^U, \emptyset, C, a_1(j)) = \rho(H^U, \emptyset, C, a_2(j)) = ... = \rho(H^U, \emptyset, C, a_{k+1}(j))$$

for any $1 \leq j \leq p-2$, $b_1(j), b_2(j)..., b_{k+1}(j)$ shares the same neighbors, therefore

$$\rho(H^U, \emptyset, C, b_1(j)) = \rho(H^U, \emptyset, C, b_2(j)) = ... = \rho(H^U, \emptyset, C, b_{k+1}(j))$$

Thus it follows from Claim 5.2 in [12] that the first vertex selected by our algorithm belongs to $\{b_1(1), b_2(1)..., b_{k+1}(1)\}$, and the densest subgraph is the entire neighborhood.

W.L.O.G. Suppose the first vertex selected by our algorithm is $b_1(1)$. Now, since the edges between U_1 and W are only covered once, so the maximum density of $b_2(1), b_3(1), ..., b_{k+1}(1)$ remains the highest. Therefore, $b_2(1), b_3(1), ..., b_{k+1}(1)$ are successively selected in the following iterations. □

At the end of the $k+1$-th iteration, the situation becomes somewhat simpler. The edges between $b_2(1), b_3(1), ..., b_{k+1}(1)$ with A, W and U_1 are already spanned, all edges between W and U_1, between A and W , between A and U_1 are covered $k+1$ times and are thus been taken care of. Inductively, it follows by arguments similar to the above that

Claim 13. *For $1 \leq j \leq p-2$, in the $j+i$-th iteration, where $1 \leq i \leq k+1$, the vertex $b_i(j)$ is chosen by our algorithm and all the edges adjacent edges are added to the spanner.*

Combined with 12, we conclude:

Lemma 14. *On graph G_k, the approximation of our algorithm is $O(\log q)$=$O(\log k)$*

Now, we take k to be $\frac{q}{\log q}$. Then our algorithm takes $O(q^2)$ edges and optimal solution takes $O(\frac{q^2}{\log q})$. Then since every vertex has more than k degrees in this graph, we immediately have:

Lemma 15. *By plugging G_k into the high degree subgraph of the example described in 5.1, our algortihm gives $O(\log \frac{km}{n})$ approximation ratio on this graph, where $k = \sqrt{\frac{n}{\log n}}$.*

Now We are going to generalize this example for both $k > \sqrt{\frac{n}{\log n}}$ and $k < \sqrt{\frac{n}{\log n}}$ cases.

5.3 Construction of Tight Example for General k

5.3.1 A Fact when $k >$ poly(m/n)

Before we get into construction, let us first make a simple observation.

Lemma 16. *If $k >$ poly(m/n), then k is upper bounded by $\tilde{O}(\sqrt{n})$*

Proof. Since $|E_{low}| \geq |V_{low}|$ and $|E_{high}| \geq \frac{1}{2} \cdot (k+1) \cdot |V_{high}| \geq \frac{1}{2} \cdot (k+1) \cdot (n - |V_{low}|)$. Therefore we have a lower bound for the total number of edges in the graph,

$$m = |E_{high}| + |E_{low}| \geq \frac{1}{2} \cdot (k+1) \cdot n - \frac{1}{2} \cdot (k-1) \cdot |V_{low}|$$

Let $k > \frac{1}{\beta}(\frac{m}{n})^\alpha$, α could be arbitrarily large, β could be arbitrarily small. Then, $m < \beta k^{\frac{1}{\alpha}} n$. Thus we have

$$|V_{low}| > (1 - 2\beta \cdot \frac{k^{\frac{1}{\alpha}}}{k}) \cdot n$$

At the mean time,

$$k < |V_{high}| = n - |V_{low}| < 2\beta \cdot \frac{k^{\frac{1}{\alpha}}}{k} \cdot n$$
$$\Rightarrow k < (2\beta n)^{\frac{1}{2-\frac{1}{\alpha}}} = O(n^{\frac{1}{2-\epsilon}})$$

Therefore, when $k > poly(\frac{m}{n})$, k is upper-bounded by $\tilde{O}(\sqrt{n})$ □

5.3.2 Construction of General Counterexamples

For $k > \sqrt{\frac{n}{\log n}}$, let the number of vertices in the low-degree subgraph be $n_l = O(n)$, the number of edges in low-degree subgraph be $m_l = O(n)$, the number of vertices in high-degree subgraph be $\sqrt{n} \cdot log^{\alpha} n$, where α is a parameter, the number of edges in high-degree subgraph be $n \cdot log^{2\alpha} n$. Then $m = O(n \cdot log^{2\alpha} n)$. Therefore,

$$\log \frac{m}{n} = O(\log \log n)$$

By directly plugging G_k described in 5.1 in the high-degree subgraph and make k to be:

$$k = \frac{\sqrt{n} \cdot log^{\alpha} n}{\log(\sqrt{n} \cdot log^{\alpha} n)} = O(\sqrt{n} \cdot log^{\alpha-1} n)$$

Lemma 17. *Our algorithm gives $O(log\frac{km}{n})$ approximation on this graph*

Proof. Our algorithm will take:

$$|ALG| = O(n) + n \cdot \log^{2\alpha} n = O(n \cdot \log^{2\alpha} n)$$

While the optimal solution would be:

$$|OPT| = O(n) + \frac{n \cdot \log^{2\alpha} n}{\log(\sqrt{n} \cdot log^{\alpha} n)} = O(n \cdot \log^{2\alpha-1} n))$$

Therefore, the approximation ratio of our algorithm is:

$$\frac{|ALG|}{|OPT|} = O(\log n) = O(log \frac{km}{n}) >> O(\log \log n) = \log \frac{m}{n}$$

□

Note that $k = \sqrt{\frac{n}{\log n}}$ when $\alpha = 1/2$, then k gets bigger when α grows. Since k is bounded by $\tilde{O}(\sqrt{n})$ when $k > poly(\frac{m}{n})$, this family of graph is a general counterexample for the case $k > \sqrt{\frac{n}{\log n}}$.

For $k < \sqrt{\frac{n}{\log n}}$, let $n_l = O(n)$, $m_l = O(n)$, let the high-degree subgraph described in 5.1 be x duplication of G_k, where each G_k contains $\sqrt{\frac{n \log n}{x}}$ vertices, therefore, $n_h = \sqrt{\frac{n \log n}{x}} \cdot x = \sqrt{x \cdot n \log n}$, $m_h = \frac{n \log n}{x} \cdot x = n \log n$. Take k to be:

$$k = \frac{\sqrt{\frac{n \log n}{x}}}{\log \sqrt{\frac{n \log n}{x}}} = \sqrt{\frac{n}{x \log n}}$$

Then $m = O(n \log n)$, $n = O(n)$,

$$\log \frac{m}{n} = O(\log \log n)$$

Lemma 18. *Our algorithm gives $O(log\frac{km}{n})$ approximation on this graph*

Proof. Our algorithm will take:

$$|ALG| = O(n) + O(n \log n) = O(n \log n)$$

While the optimal solution will take:

$$|OPT| = O(n) + O(\frac{n \log n}{\log \sqrt{x \cdot n \log n}}) = O(n)$$

So the approximation ratio of our algorithm is:

$$\frac{|ALG|}{|OPT|} = O(\log n) = O(log\frac{km}{n}) >> O(\log\log n) = \log\frac{m}{n}$$

□

Note that $k = \sqrt{\frac{n}{\log n}}$ when $x = 1$, which degenerates to our counterexample in 5.2. By making x bigger, k turns smaller. Therefore, this family of graph is a general counterexample for the case $k < \sqrt{\frac{n}{\log n}}$.

This construction builds a general counterexample for our algorithm and implies the analysis of $O(\log\frac{km}{n})$ approximation ratio bound is tight.

Lemma 19. *Our algorithm gives $O(log\frac{km}{n})$ approximation on the family of graph described in section 5*

References

[1] Baruch Awerbuch and David Peleg. Online tracking of mobile users. *Journal of the ACM (JACM)*, 42(5):1021–1058, 1995.

[2] Greg Bodwin, Michael Dinitz, and Yasamin Nazari. Vertex Fault-Tolerant Emulators. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:22, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[3] 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.

[4] Greg Bodwin, Michael Dinitz, and Caleb Robelle. Optimal vertex fault-tolerant spanners in polynomial time. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, 2021.

[5] Greg Bodwin, Michael Dinitz, and Caleb Robelle. Partially optimal edge fault-tolerant spanners. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 3272–3286. SIAM, 2022.

[6] 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.

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

[8] Michael Dinitz and Robert Krauthgamer. Directed spanners via flow-based linear programs. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, STOC '11, page 323–332, New York, NY, USA, 2011. Association for Computing Machinery.

[9] Michael Dinitz and Robert Krauthgamer. Fault-tolerant spanners: better and simpler. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 169–178, 2011.

[10] 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.

[11] Michael Elkin, Yuval Emek, Daniel A Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 494–503, 2005.

[12] Guy Kortsarz and David Peleg. Generating sparse 2-spanners. *Journal of Algorithms*, 17(2):222–236, 1994.

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

[14] David Peleg and Jeffrey D Ullman. An optimal synchronizer for the hypercube. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 77–85, 1987.

[15] Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90, 2004.

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

A Equivalence of Vertex and Edge Fault-Tolerance

Now we prove edge fault setting is equivalent to vertex fault setting for stretch 2.

Definition 3. A subgraph G' of a graph $G = (V, E)$ is k-edge fault tolerant (k-EFT) 2-spanner of G if

$$d_{G' \setminus F}(u, v) \leq 2 \cdot d_{G \setminus F}(u, v)$$

for all $F \subseteq E$ with $|F| \leq k$ and $u, v \in V$.

Theorem 20. *Let $G = (V, E)$, and let $G' = (V, E')$ be a subgraph of G. Then G' is a k-VFT 2-spanner of G if and only if G' is a k-EFT 2-spanner.*

Proof. Let G' be a k-VFT 2-spanner of G. Given arbitrary edge fault set $F \subseteq E$ with $|F| \leq k$. Let $(u, v) \in E$, consider any edge $e = (p, q)$ that lies on the shortest path between u and v in $G \setminus F$. If $e \in E' \setminus F$, then $d_{G' \setminus F}(p, q) = 1$, otherwise, by 1, e is covered $k + 1$ times by E', so there is at least one path of length 2 between p and q in $E' \setminus F$. Thus $d_{G' \setminus F}(p, q) \leq 2$. Therefore, the shortest path between u and v is distorted by at most 2, *i.e.* $d_{G' \setminus F}(u, v) \leq 2 \cdot d_{G \setminus F}(u, v)$, which indicates G' is a k-EFT 2-spanner of G.

Let G' be a k-EFT 2-spanner of G. For the sake of contradiction, assume $\exists (u, v) \in E$ such that $(u, v) \notin E'$ and (u, v) is covered at most k times by E', denote the vertices that covers (u, v) using edges in E' by $w_1, w_2, ..., w_x$, where $x \leq k$. Then if we take edge fault set $F \subseteq E$ to be $\{(u, w_i) : i \in [x]\}$, in the remaining graph $G' \setminus F$ there is no $u - v$ path, while in $G \setminus F$ the edge (u, v) exists. Thus G' is not a k-EFT 2-spanner of G, giving the contradiction. Therefore, for every $(u, v) \in E$, either $(u, v) \in E'$ or (u, v) is covered at least $k + 1$ times by E', which indicates G' is a k-VFT 2-spanner of G. □