A Polynomial Time Algorithm for Almost Optimal Vertex Fault Tolerant Spanners

Udit Agarwal
udit@utexas.edu
University of Texas at Austin
Austin, Texas

February 27, 2020

Abstract

We present the first polynomial time algorithm for the \( f \) vertex fault tolerant spanner problem, which achieves almost optimal spanner size. Our algorithm for constructing \( f \) vertex fault tolerant spanner takes \( O(k \cdot n \cdot m^2 \cdot W) \) time, where \( W \) is the maximum edge weight, and constructs a spanner of size \( O(n^{1+1/k} f^{1-1/k} \cdot (\log n)^{1-1/k}) \). Our spanner has almost optimal size and is at most a \( \log n \) factor away from the upper bound on the worst-case size. Prior to this work, no other polynomial time algorithm was known for constructing \( f \) vertex fault tolerant spanner with optimal size.

Our algorithm is based on first greedily constructing a hitting set for the collection of paths of weight at most \( k \cdot w(u,v) \) between the endpoints \( u \) and \( v \) of an edge \((u,v)\) and then using this set to decide whether the edge \((u,v)\) needs to be added to the growing spanner.

1 Introduction

In this paper we study an efficient construction of \( f \) vertex fault tolerant vertex spanners in the sequential model. A spanner \( H \) is a subgraph of a graph \( G = (V,E) \) such that it preserves distances between all pairs \((u,v)\) by a factor of at most \( k \), i.e. \( d_H(u,v) \leq k \cdot d_G(u,v) \). However a subgraph \( H \) is called a \( f \) vertex fault tolerant spanner of \( G \) if for any set of at most \( f \) vertices, \( F \subset V \), the resulting subgraph \( H \setminus F \) is a \( k \)-spanner of \( G \setminus F \).

Spanners were first introduced by Peleg and Schäffer [13] and Peleg and Ullman [14], and has been extensively studied over the years (e.g. [1, 3, 6, 10, 11, 15]). In practice, spanners are mostly used in applications in the area of distributed computing. However distributed systems are prone to failures, and thus we would like a spanner for such a system to be robust to these failures, giving rise to the need for constructing fault-tolerant spanners.

This notion of fault-tolerant spanners was first introduced by Levcopoulos, Narasimhan, and Smid [12] and has been intensively studied over the years as well [5, 7, 9]. A naive way for constructing an \( f \)-fault tolerant spanner is to construct it greedily by going over the edges in increasing order of their weight. Recently Bodwin and Patel [5] showed that such an approach gives an optimal bound on the worst-case size of such spanners. This improves on the bound achieved in [4] which was the first such result based on the greedy algorithm and all prior work on fault tolerant spanners uses more involved constructions with comparatively simpler analysis (e.g. [2, 7, 8]).
However the running time of the above described greedy approach is exponential in $f$ (also mentioned in [3]) and in [4] the authors also mentions that it would be interesting to improve this exponential dependence on $f$, or perhaps to find a different fast algorithm achieving the existential bounds shown in the paper. In this paper we partly solve the problem by providing a polynomial time algorithm for constructing $f$ vertex fault tolerant spanners, with runtime polynomially dependent on the maximum edge weight, $W$.

2 Overview of the Algorithm

We first start with a brief overview of the naive greedy algorithm for constructing $f$ vertex fault tolerant spanners, which is described below.

Algorithm 1 Greedy-VFT
Input: $G = (V, E); k; f$
1: $H \leftarrow \emptyset$
2: for each $(u, v) \in E$ in order of non-decreasing edge weights do
3: if there exists $F \subset V$ of size at most $f$ such that $\text{dist}_{H \setminus F}(u, v) > k \cdot w(u, v)$ then
4: add $(u, v)$ to $H$
5: end if
6: end for

The correctness of the above described algorithm is quite straightforward. However its running time is exponential in the parameter $f$ since a naive implementation of the if loop in Steps 3-5 by going over all possible subsets $F$ of $V$ of size at most $f$ and then checking the weight of the shortest path from $u$ to $v$ in the resulting graph $H\setminus F$, takes $\Omega(n^f)$ time (also noted in [3]). In this paper we present an alternate greedy algorithm for constructing $f$ vertex fault tolerant vertex spanners with polynomial running time. The running time of our algorithm is independent of the parameter $f$.

Our algorithm is still based on the greedy approach and it goes over the edges in non-decreasing order of weights. However we use an entirely different method than the one described in Algorithm 1 to decide whether the edge $(u, v)$ needs to be added to the growing spanner $H$. Our method constructs a greedy hitting set for the collection of paths from $u$ to $v$ in $H$ of weight at most $k \cdot w(u, v)$ and then decide whether to add $(u, v)$ to $H$ or not based on the size of this hitting set. Note that we do not list all paths from $u$ to $v$ in $H$ since it will require exponential time. Instead for each vertex $x$, our algorithm computes a count of the number of paths from $u$ to $v$ that passes through $x$ and has weight at most $k \cdot w(u, v)$. This allows us to compute a desired hitting set for the collection of paths from $u$ to $v$ without explicitly listing these paths. This is described in detail in Section 3.

3 Main Contributions

Polynomial Time Greedy Algorithm for Computing $f$ vertex fault tolerant Spanner. Prior to this work, all greedy approaches for constructing $f$ vertex fault tolerant spanners require exponential time [4,5]. This is the first work that presents a polynomial time algorithm for constructing $f$ vertex fault tolerant spanners for integer edge weights.
Construction of Hitting Set for a collection of paths without explicitly listing the paths. Our algorithm for constructing \( f \) vertex fault tolerant spanners requires constructing a hitting set for the collection of paths from \( u \) to \( v \) of weight at most \( k \cdot w(u, v) \) in the growing spanner \( H \) before deciding whether to add the edge \((u, v)\) to \( H \). However since explicitly listing all paths from \( u \) to \( v \) may require exponential time, we present an alternate way for constructing the hitting set by computing for each node \( x \), the number of paths from \( u \) to \( v \) that passes through \( x \).

4 Description of the Algorithm

In this Section we give a detailed description of our algorithm for constructing \( f \) vertex fault tolerant spanners. As noted in Section 2, our algorithm is also based on the greedy approach and it goes over the edges \((u, v)\) in non-decreasing order of weight. Algorithm 2 gives the pseudocode of our algorithm.

Algorithm 2 POLY-VFT
\[
\begin{align*}
\text{Input: } & G = (V, E); \ k; \ f \\
1: & H \leftarrow \phi \\
2: & \text{for each } (u, v) \in E \text{ in order of non-decreasing edge weights do} \\
3: & \quad \text{Let } C \text{ be the collection of paths from } u \text{ to } v \text{ of weight at most } k \cdot w(u, v) \text{ (we do not explicitly compute } C). \ \\
4: & \quad \text{Construct hitting set } Q \text{ for the collection } C \text{ using Algorithm 3 described in Section 5} \\
5: & \quad \text{if } |Q| \leq f \cdot \log n \text{ then} \\
6: & \quad \quad \text{add } (u, v) \text{ to } H \\
7: & \quad \text{end if} \\
8: & \text{end for}
\end{align*}
\]

We now give a step by step description of Algorithm 2. The algorithm goes over all the edges \((u, v)\) in the non-decreasing order of weights in the for loop in Steps 2-8. In each iteration of the for loop, the algorithm first construct a hitting set \( Q \) for the collection of paths from \( u \) to \( v \) of weight at most \( k \cdot w(u, v) \) using the greedy hitting set algorithm described in Section 5. Then in Steps 5-7, the algorithm adds the edge \((u, v)\) to the growing spanner \( H \) if the size of the hitting set \( Q \) is at most \( f \cdot \log n \).

4.1 Correctness

We now establish the correctness of Algorithm 2. We start with establishing that \( H \) is indeed a \( f \) vertex fault tolerant spanner (Lemma 4.1). We then establish the polynomial running time of Algorithm 2 (Lemma 4.2). We conclude with establishing a bound on the size of \( H \) (Lemmas 4.4-4.7).

Lemma 4.1. The subgraph \( H \) constructed by Algorithm 2 is a \( f \) vertex fault tolerant spanner.

Proof. Let \( H' \) be a \( f \) vertex fault tolerant spanner for the graph \( G \) and let \( H \) be the subgraph constructed by Algorithm 2. Let \((u, v)\) be a minimum weight edge that belongs to the set \( H' \setminus H \). Let \( C \) be the collection of paths from \( u \) to \( v \) in \( H \) of weight at most \( k \cdot w(u, v) \). Since \((u, v)\) was
not added to $H$ in Steps 5-7, it implies that $Q$ has size more than $f \cdot \log n$. From Lemma 5.1, any optimal set $F$ that hits every path in the collection $C$ should have size greater than $\frac{|Q|}{\log n} > f$. Thus there will always exist at least one path from $u$ to $v$ in $H$ of weight at most $k \cdot w(u,v)$, irrespective of the set $F$ of nodes that is removed from the graph such that $|F| \leq f$.

In a similar fashion, we can argue about the rest of the edges in the set $H' \setminus H$. Thus $H$ is a $f$ vertex fault tolerant spanner.

We now establish the polynomial runtime of Algorithm 2.

**Lemma 4.2.** Algorithm 2 constructs the spanner $H$ in $O(k \cdot n \cdot m^2 \cdot W)$ time, where $W$ is the maximum integer edge weight.

**Proof.** The for loop in Steps 2-8 runs for total $m$ iterations. And each iteration of the for loop takes $O(k \cdot n \cdot m \cdot W)$ running time, since Step 4 takes $O(k \cdot n \cdot m \cdot W)$ time by Lemma 5.2 and Steps 5-7 take $O(1)$ time. 

We finally conclude that Algorithm 2 constructs an almost optimal size $f$ vertex fault tolerant spanner by establishing that $H$ has almost optimal size. Our proof is based on the proof given in [5] to establish the upper bound on the worst-case size of a $f$ vertex fault tolerant spanner. We describe the complete proof here for completeness.

4.1.1 Bound on size of $H$

We start with the following definition of a blocking set, described in [5].

**Definition 4.3** (Blocking Set [5]). Given a graph $G = (V, E)$, a $k$-blocking set for $G$ is a set $B \subseteq V \times E$ such that:

(a) every $(v, e) \in B$ has $v \notin e$, and

(b) for every cycle $C$ in $G$ on $\leq k$ edges, there exists $(v, e) \in B$ such that $v, e \in C$.

We now give a upper bound on the size of the blocking set $B$ of the spanner $H$ constructed by Algorithm 2. The proof is similar to the proof of Lemma 3 in [5].

**Lemma 4.4.** The spanner $H$ returned by Algorithm 2 with parameters $k, f$ has a $(k + 1)$-blocking set of size at most $(f \log n) \cdot |E(H)|$.

**Proof.** For each edge $e = (u, v) \in H$, let $Q_e$ be the hitting set constructed in Step 4 for the collection of paths from $u$ to $v$ of weight at most $k \cdot w(u,v)$, when $(u,v)$ was added to $H$. Let $B = \{(x, e) | e \in E(H), x \in Q_e\}$. Clearly $|B| \leq f \cdot \log n \cdot |E(H)|$ since an edge $e$ is added to $H$ in Steps 5-7 only if $|Q_e| \leq f \cdot \log n$.

We now show that $B$ is a $(k + 1)$-blocking set. Let $C$ be a cycle in $H$ with at most $k + 1$ edges and let $(u, v)$ be the edge which was added the last to the spanner $H$ among all the edges in $C$. Since the path from $u$ to $v$ through the cycle $C$ has length at most $k$, the weight of this path is at most $k \cdot w(u,v)$ (since $(u, v)$ has the largest weight in $C$) and thus this path must be in the collection $C$. Since $Q_e$ is a hitting set for the paths in $C$, there exists $x \in Q$ that also belongs to $C \setminus \{u, v\}$ and by construction of $B$, $(x, e) \in B$. This shows that $B$ is indeed a $(k + 1)$-blocking set.
We adapt the following lemma from [5] (Lemma 4) to establish an upper bound on the size of spanner \( H \) (Theorem 4.6).

**Lemma 4.5** ([5]). Let \( H \) be any graph on \( n \) nodes and \( m \) edges, let \( f = o(n) \) be a parameter and let \( f' = f \cdot \log n \), and suppose \( H \) has a \((k+1)\)-blocking set \( B \) of size \(|B| \leq f' \cdot |E(H)|\). Then \( H \) has a subgraph on \( O(n/f') \) nodes, \( \Omega(m/f'^2) \) edges, and girth \( > k + 1 \).

The following theorem along with Corollary 4.7 from [5] establishes an upper bound on the worst-case size of \( H \).

**Theorem 4.6** ([5]). Let \( b(n,k) \) be the maximum possible number of edges in a graph on \( n \) nodes and girth \( > k \). Then any graph \( H \) on \( n \) nodes returned by Algorithm 2 with parameters \( f', k \) satisfies

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

**Corollary 4.7** ([5]). For any graph \( H \) on \( n \) nodes returned by Algorithm 2 with parameters \( f', 2k-1 \), we have

\[
|E(H)| = O(n^{1+1/k} f'^{-1/k})
\]

And since \( f' = f \cdot \log n \), we have \(|E(H)| = O(n^{1+1/k} \cdot f'^{1-1/k} \cdot (\log n)^{1-1/k})\).

## 5 Hitting Set Algorithm

In this section we describe an algorithm to implement Step 4 of Algorithm 2. The goal is to construct a small hitting set for the collection \( C \) of paths from vertex \( u \) to \( v \) of weight at most \( k \cdot w(u, v) \). Note that we do not explicitly construct this collection \( C \) since it may require exponential time.

Our algorithm follows a greedy approach to construct the hitting set \( Q \). It proceeds in iterations: in each iteration, it first uses a dynamic programming procedure to compute, for each vertex \( x \), the number of paths from \( u \) to \( v \) that pass through \( x \) in \( C \) (call this value \( \text{count}_x \)). It then picks the vertex \( y \), that has the maximum count value, in set \( Q \) (paths through \( y \) are not considered in the collection \( C \) in future iterations). Algorithm 3 gives the pseudocode of our hitting set algorithm.

**Algorithm 3** Hitting-Set Construction

**Input:** \( G = (V,E); \ k, u, v \)

1: \( Q \leftarrow \emptyset \)
2: Compute \( \text{count}_x \) values for each \( x \in V \) using the algorithm described in Section 5.1
3: while there exists \( x \in V \) with \( \text{count}_x > 0 \) do
4: \( y \) be the node with max \( \text{count} \) value.
5: add \( y \) to \( Q \).
6: Re-compute \( \text{count}_x \) values for each \( x \in V \) using the algorithm described in Section 5.1
7: end while

The correctness of Algorithm 3 is quite straightforward. Lemma 5.1 establishes a relationship between the size of an optimal hitting set and the constructed set \( Q \) and Lemma 5.2 establishes the running time of the algorithm.
Lemma 5.1 ((Folklore)). A hitting set $Q$ constructed by a greedy algorithm has size at most $\log n \cdot \text{OPT}$, where OPT is the size of an optimal solution.

Lemma 5.2. Algorithm 3 runs in $O(k \cdot n \cdot m \cdot W)$ time.

Proof. Step 2 takes $O(k \cdot m \cdot W)$ time (Corollary 5.5). Each iteration of the while loop in Steps 3-7 takes $O(k \cdot m \cdot W)$ time since Step 5 takes $O(1)$ time and Step 6 takes $O(k \cdot m \cdot W)$ time. The lemma follows since the while loop runs for at most $n$ iterations.

5.1 Computing $\text{count}_x$ values

In this section we describe a dynamic programming algorithm to compute $\text{count}_x$ values for each vertex $x$, where $\text{count}_x$ is the number of paths in the collection $\mathcal{C}$ that passes through $x$ (excluding the paths that are already covered by the already computed hitting set $Q$). Here we are given the endpoints $u$ and $v$ of all the paths in this collection $\mathcal{C}$.

Our algorithm follows a three-phase strategy: first for each $x \in V$, we compute the number of paths starting from $u$ and ending at $x$ of weight $wt$, where $1 \leq wt \leq k \cdot w(u, v)$ and then similarly we compute for each $x \in V$, the number of paths starting at $x$ and ending at $v$ of weight $wt$ for $1 \leq wt \leq k \cdot w(u, v)$. We then combine these values to obtain the $\text{count}_x$ values for each $x \in V$.

Let $\text{count}_{u, x, wt}$ refers to the number of paths in the collection $\mathcal{C}$ (excluding the paths covered by the already computed vertices in $Q$) that starts from $u$ and ends at $x$ and has weight $wt$. Similarly let $\text{count}_{x, v, wt}$ refers to the number of paths in the collection $\mathcal{C}$ (excluding the paths covered by the already computed vertices in $Q$) that starts from $x$ and ends at $v$ and has weight $wt$.

5.1.1 Computing $\text{count}_{u, x, wt}$ values

We follow a dynamic programming approach to compute $\text{count}_{u, x, wt}$ values for each $x \in V$ and $1 \leq wt \leq k \cdot w(u, v)$. Algorithm 4 describes the pseudocode of our algorithm.

Algorithm 4 \text{COMPUTE-\text{count}_{u, x, wt}}

Input: $G = (V, E); k; u, v, Q$: current vertices in hitting set
1: for each $(u, x) \in E$ do
2: \hspace{1em} $\text{count}_{u, x, w(u, x)} \leftarrow 1$
3: end for
4: for $1 \leq wt \leq k \cdot w(u, v)$ do
5: \hspace{1em} for each $(x, y) \in E$ do
6: \hspace{2em} if $\text{count}_{u, x, wt} > 0$ then
7: \hspace{3em} $\text{count}_{u, y, wt + w(x, y)} \leftarrow \text{count}_{u, x, wt} + \text{count}_{u, y, wt + w(x, y)}$
8: \hspace{2em} end if
9: \hspace{1em} end for
10: end for

The correctness of the above algorithm is quite straightforward. We now establish the running time of the above algorithm.
Lemma 5.3. Algorithm 4 computes \( \text{count}_{u,x,wt} \) values for each \( x \in V, 1 \leq wt \leq k \cdot w(u,v) \) in \( O(k \cdot m \cdot W) \) time.

Proof. The inner for loop in Steps 5-9 takes \( O(m) \) time per iteration of the outer for loop. The lemma follows since the outer for loop runs for \( k \cdot W \) iterations. \qed

5.1.2 Computing \( \text{count}_{x,v,wt} \) values

Similar to the algorithm described in the previous section, we can compute \( \text{count}_{x,v,wt} \) values using a dynamic programming approach in \( O(k \cdot m \cdot W) \) time. This leads to the following lemma.

Lemma 5.4. For each \( x \in V, 1 \leq wt \leq k \cdot w(u,v) \), the \( \text{count}_{x,v,wt} \) values can be computed in \( O(k \cdot m \cdot W) \) time.

5.1.3 Computing \( \text{count}_{x} \) values

In this section we describe a simple algorithm to combine the \( \text{count}_{x} \) values using the \( \text{count}_{u,x,wt} \) and \( \text{count}_{x,v,wt} \) values computed in the previous sections.

Let \( \text{sum}_{u,x,wt} \) denote the sum of \( \text{count}_{u,x,wt} \) values for \( 1 \leq wt' \leq wt \). We can compute these values for each \( x \in V, 1 \leq wt \leq k \cdot w(u,v) \), in \( O(k \cdot n \cdot W) \) time. Then for each \( x \in V \), \( \text{count}_{x} \) is the sum of \( \text{sum}_{u,x,wt} \) and \( \text{count}_{x,v,wt} \) for each \( 1 \leq wt \leq k \cdot w(u,v) - 1 \). This can be done in additional \( O(k \cdot W) \) time per \( x \in V \). This leads to the following lemma.

Lemma 5.5. Given \( \text{count}_{u,x,wt} \) and \( \text{count}_{x,v,wt} \) values, for each \( x \in V \) \( \text{count}_{x} \) values can be computed in additional \( O(k \cdot n \cdot W) \) time.

The following corollary follows from Lemmas 5.3, 5.4, 5.5.

Corollary 5.6. For given endpoints \( u \) and \( v \), the \( \text{count}_{x} \) values for each \( x \in V \) can be computed in \( O(k \cdot m \cdot W) \) time.

6 Conclusion

In this paper we present a first polynomial time algorithm for computing \( f \) vertex fault tolerant spanners of almost optimal size. The size of our spanner is at most a \( \log n \) factor away from the known optimal bound \[5\].

References

[1] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993.

[2] G. Ausiello, A. Ribichini, P. G. Franciosa, and G. F. Italiano. Computing graph spanners in small memory: fault-tolerance and streaming. *Discrete Mathematics, Algorithms and Applications*, 2(04):591–605, 2010.
[3] S. Baswana, T. Kavitha, K. Mehlhorn, and S. Pettie. Additive spanners and \((\alpha, \beta)\)-spanners. *ACM Transactions on Algorithms (TALG)*, 7(1):1–26, 2010.

[4] G. Bodwin, M. Dinitz, M. Parter, and V. V. Williams. Optimal vertex fault tolerant spanners (for fixed stretch). In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1884–1900. SIAM, 2018.

[5] G. Bodwin and S. Patel. A trivial yet optimal solution to vertex fault tolerant spanners. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 541–543, 2019.

[6] S. Chechik. New additive spanners. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 498–512. SIAM, 2013.

[7] S. Chechik, M. Langberg, D. Peleg, and L. Roditty. Fault tolerant spanners for general graphs. *SIAM Journal on Computing*, 39(7):3403–3423, 2010.

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

[9] M. Dinitz and R. 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] M. Elkin and D. Peleg. \((1+\tilde{\alpha},\beta)\)-spanner constructions for general graphs. *SIAM Journal on Computing*, 33(3):608–631, 2004.

[11] A. Filtser and S. Solomon. The greedy spanner is existentially optimal. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 9–17, 2016.

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

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

[14] D. Peleg and J. D. Ullman. An optimal synchronizer for the hypercube. *SIAM Journal on Computing*, 18(4):740–747, 1989.

[15] M. Thorup and U. Zwick. Spanners and emulators with sublinear distance errors. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 802–809, 2006.