# Improved Parallel Algorithms for Spanners and Hopsets

Gary L. Miller
Carnegie Mellon University
glmiller@cs.cmu.edu

Richard Peng
MIT
rpeng@mit.edu

Adrian Vladu
MIT
avladu@mit.edu

Shen Chen Xu
Carnegie Mellon University
shenchex@cs.cmu.edu

## ABSTRACT

We use exponential start time clustering to design faster parallel graph algorithms involving distances. Previous algorithms usually rely on graph decomposition routines with strict restrictions on the diameters of the decomposed pieces. We weaken these bounds in favor of stronger local probabilistic guarantees. This allows more direct analyses of the overall process, giving:

- Linear work parallel algorithms that construct spanners with $O(k)$ stretch and size $O(n^{1+1/k})$ in unweighted graphs, and size $O(n^{1+1/k} \log k)$ in weighted graphs.
- Hopsets that lead to the first parallel algorithm for approximating shortest paths in undirected graphs with $O(m \operatorname{poly} \log n)$ work.

## 1. INTRODUCTION

Graph decompositions are widely used algorithmic routines. They partition the graph to enable divide-and-conquer algorithms. One form that has proven to be particularly useful is the low diameter decomposition: the decomposed pieces should have small diameter, while few edges have endpoints in different pieces. Variants of the low diameter decomposition are used in algorithms for spanners [Coh98], distance oracles [TZ05], and low stretch embeddings [AKPW95, Bar96, CMP+14].

Early applications of the low diameter decomposition include distributed algorithms by Awerbuch [Awe85], and low-stretch spanning trees by Alon et al. [AKPW95]. Further study of low stretch tree embeddings led to a probabilistic decomposition routine by Bartal [Bar96]. On an unweighted graph, this decomposition partitions the graph so that only a $\beta$ fraction of the edges are cut, and the resulting pieces have diameter $O(\beta^{-1} \log n)$.

The development of parallel algorithms for finding tree embeddings [BGK+14] led to a parallel low diameter clustering routine using exponential start times [MPX13]. This routine then led to algorithms that generate tree embeddings suitable for a variety of applications [CMP+14]. The clustering algorithm itself has properties suitable for reducing the communication required in parallel connectivity algorithms [SDB14]. This suggests that exponential start time clusterings have a variety of other applications in graph algorithms.

Graph decomposition routines are often invoked hierarchically, leading to many levels, each refining the output of the previous one. The $O(\log n)$ discrepancy between the probability of edges being cut and diameters of pieces in standard low diameter decomposition could then accumulate through the levels. To address this issue, recent algorithms using low diameter decompositions usually require stronger properties at intermediate steps.

In this paper, we give an alternate approach based on the probabilistic guarantees of the exponential start time clustering. We regard the multiple levels as independent events, and analyze the output probabilistically in each locality. This weakens the interactions between the levels, while still allowing us to analyze the final outcome using probabilistic methods. We apply this method to several classical graph problems involving distances.

Spanners are sparse subgraphs that approximate distances in the original graph. We show that one round of exponential start time clustering augmented with a few edges leads to spanners. This algorithm extends to the weighted setting by bucketing the edges by weights, and then clustering them hierarchically. This leads to an overhead of $O(\log k)$ in the size compared to the optimal construction where $k$ is the stretch factor, and $O(\log U)$ in depth, where $U$ is the ratio between the maximum and minimum edge weights.

THEOREM 1.1. *There exists an algorithm that given as input a graph $G$ and parameter $k \geq 1$, finds with high probability a subgraph $H$ in which shortest path distances are preserved up to a factor of $O(k)$ (i.e. $H$ is a $O(k)$-spanner of $G$). If $G$ is an unweighted graph, then $H$ has expected size $O(n^{1+1/k})$ and is computed in $O(\log n \log^* n)$ depth with $O(m)$ work. If $G$ is a weighted graph with ratio of maximum and minimum edge weights bounded by $U$, $H$ has expected size $O(n^{1+1/k} \log k)$ and is computed in $O(\log U \log n \log^* n)$ depth with $O(m)$ work.*

Closely related to spanners are hopsets, which do not limit the edge count, but aim to reduce the number of edges in the shortest paths. These objects are crucial for speeding up parallel algorithms for (approximate) shortest paths [KS97,


*SPAA'15,* June 13–15, 2015, Portland, OR, USA.

ACM 978-1-4503-3588-1/15/06 ...$15.00.
DOI: http://dx.doi.org/10.1145/2755573.2755574.

Coh00]. Using the exponential start time clustering, we construct hopsets which lead to the first parallel algorithm for approximating shortest paths in undirected graphs with $O(m \operatorname{poly} \log n)$ work. Here, our key idea is to employ backward analysis and analyze the algorithm with respect to a single (unknown) optimal $s$-$t$ path. We show that in expectation this path is not cut in too many places by the decomposition scheme, and use this to bound the overall distortion.

THEOREM 1.2. *There exists an algorithm that given as input an undirected, non-negatively weighted graph $G$, and parameters $\alpha, \epsilon \in (0,1)$, preprocesses the graph in $O(m\epsilon^{-2-\alpha} \log^{3+\alpha} n)$ work and $O\left(n^{\frac{4+\alpha}{4+2\alpha}} \epsilon^{-1-\alpha} \log^2 n \log^* n\right)$ depth, so that for any vertices $s$ and $t$ one can return a $(1+\epsilon)$-approximation to the $s-t$ shortest in $O(m\epsilon^{-1-\alpha})$ work and $O\left(n^{\frac{4+\alpha}{4+2\alpha}} \epsilon^{-2-\alpha}\right)$ depth.*

The paper is organized as follows: Section 2 reviews some standard notions and compares our results with related works. In Section 3, we describe our spanner construction for both unweighted and weighted graphs. We then describe our hopset algorithm for unweighted graphs in Section 4, and extend it to the weighted setting in Section 5.

Although the probabilistic analysis allows us to decouple some of the levels, mild dependencies between the levels remain. Such dependencies result in terms of $O(\log k)$ and $O(\log^2 n)$ in our results for spanners and hopsets respectively. A promising direction for improvements is to construct the clusterings on different levels in a dependent manner. Picking the randomness across levels from the same source could allow more streamlined analysis of the overall process.

## 2. BACKGROUND AND RELATED WORKS

We consider a graph $G = (V, E, w)$ with $|V| = n$, $|E| = m$ and edge weights/lengths $w : E \to \mathbb{R}_+$. Throughout the paper we will only deal with undirected graphs with positive edge weights, so we can assume $w(e) \geq 1$ by normalizing and $w(u,v) = w(v,u)$. Furthermore, the graph is unweighted if $w(e) = 1$ for all $e \in E$. If $X$ is a subset of $V$ or $E$, we will use $G[X]$ to denote the induced subgraph of $G$ on $X$. If $H$ is a subgraph of $G$, we will use $G/H$ to denote the quotient graph obtained from $G$ after contracting the connected components of $H$ into points, removing self-loops and merging parallel edges (by keeping the shortest edge).

The parallel performances of our algorithms are analyzed in the standard PRAM model. The longest sequence of dependent operations is known as depth, while the total number of operations performed is termed work. In practice, the abilities of algorithms to parallelize are often limited by the number of processors. For instance, a common assumption in the MapReduce model is that the number of processors is $n^\delta$ for some small $\delta$ [KSV10]. In such settings, an algorithm will fully parallelize as long as the depth is less than $n^{1-\delta}$. As a result, it is more important to reduce work in order to obtain speed-ups over sequential algorithms.

### 2.1 Exponential Start Time Clustering

We start by formalizing the key routine in this paper, a graph decomposition routine which we call Exponential Start Time Clustering. It generates a partition of $V$ into subsets $X_1, \cdots, X_k$, and a center $c_i$ for each $X_i$. It also outputs a spanning tree for each cluster rooted at its center. For convenience, if $v \in X_i$, we use $c(v)$ to denote $c_i$. We use a routine from [MPX13].

**Algorithm 1** Exponential Start Time Clustering

ESTCLUSTER$(G, \beta)$

Input: Graph $G = (V, E, w)$, parameter $0 < \beta < 1$.

Output: Decomposition of $G$.

1: For each vertex $u$, pick $\delta_u$ independently from the exponential distribution $\exp(\beta)$.
2: Create clusters by assigning each $v \in V$ to $u = \arg\min_{u \in V}\{\text{dist}(u,v) - \delta_u\}$, if $v = u$ we let it be the center its cluster.
3: Return the clusters along with a spanning tree on each cluster rooted at its center.

In this paper we extend the algorithm to efficiently run on weighted graphs and also extend the analysis bounding the number of inter-cluster edges to more general subgraphs. The following lemma gives bounds on the run time and cluster diameter for the weighted case.

LEMMA 2.1. *(Theorem 1.2 from [MPX13]) Given a weighted graph $G = (V, E, w)$ where $|V| = n$, $|E| = m$, $w : E \to \mathbb{Z}_+$ with $\min_{e \in E} w(e) = 1$, ESTCLUSTER$(G, \beta)$ generates a set of disjoint clusters $\cup_i X_i = V$. The diameter of each $X_i$ is certified by a spanning tree on $X_i$, which has diameter at most $\frac{k}{\beta} \log n$ with probability at least $1 - 1/n^{k-1}$, for any $k \geq 1$. This computation takes $O(\beta^{-1} \log n \log^* n)$ depth with high probability and $O(m)$ work.*

We discuss the efficient implementation of the ESTCLUSTER routine in Appendix A, as well as the effect different models of parallelism have on the depth. An analysis of the resulting decomposition in unweighted graphs is in Section 4 of [MPX13], and it extends immediately to weighted graphs. Our spanner algorithm requires a stronger variant of the edge cutting guarantee which we state below and prove in Appendix A.

Let $G$ be a weighted graph, a ball centered at $c$ of radius $r$ is defined as $B(c, r) = \{v \in V \mid d(v, c) \leq r\}$. The center $c$ may either be a vertex or the midpoint of an edge. We can show that balls with small radius do not intersect with too many clusters.

LEMMA 2.2. *The probability that a ball of radius $r$ intersects at least $k$ clusters from* ESTCLUSTER *is at most $\gamma^{k-1}$ where $\gamma = 1 - \exp(-2r\beta)$.*

COROLLARY 2.3. *An edge $e$ with weight $w(e)$ is cut in the clustering produced by* ESTCLUSTER *with probability at most $1 - \exp(-\beta \cdot w(e)) < \beta \cdot w(e)$.*

### 2.2 Spanners

A subgraph $H$ of $G$ is said to be a $k$-spanner, if for every $u, v \in V$, we have $\text{dist}_H(u,v) \leq k \cdot \text{dist}_G(u,v)$, where $k$ is also called the stretch factor. Notice that it is sufficient to prove the stretch bound for endpoints of every edge. It is known that for any integer $k \geq 1$, any undirected graph with $n$ vertices admits $(2k-1)$-spanners with $O(n^{1+1/k})$ edges, and this is essentially the best tradeoff between sparsity and

**weighted graphs**

| multiplicative distortion | Size | Work | Parallel depth | Notes |
|---|---|---|---|---|
| $2k-1$ | $\frac{1}{2}n^{1+1/k}$ | $O(mn^{1+1/k})$ | $O(n^{1+1/k})$ | [ADD+93] |
| $2k-1$ | $O(kn^{1+1/k})$ | $O(km)$ | $O(\log n)$ | [BS07] |
| $2k-1$ | $O(kn^{1+1/k})$ | $O(km)$ | $O(\log n)$ | [RTZ05] |
| $O(k)$ | $O(n^{1+1/k}\log k)$ | $O(m)$ | $O(\log U \log n \log^* n)$ | new |

**unweighted graphs**

| multiplicative distortion | Size | Work | Parallel depth | Notes |
|---|---|---|---|---|
| $2k-1$ | $n^{1+1/k}$ | $O(m)$ | $O(\log^3 n)$ | [Pel00] |
| $2k-1$ | $O(kn^{1+1/k})$ | $O(km)$ | $O(\log n)$ | [BKMP10] |
| $O(2^{\log^* n}\log n)$ | $O(n)$ | $O(m\log n)$ | $O(\log n)$ | [Pet08] |
| $O(k)$ | $O(n^{1+1/k})$ | $O(m)$ | $O(\log n \log^* n)$ | new |

**Figure 1: Known results for spanners, $U$ represents the range of weights.**

stretch [PS89, TZ05]. These notions have also been studied under additive error [DHZ00, BKMP05].

A summary of efficient algorithms for constructing spanners can be found in Figure 1. Our algorithms improve upon the $O(k)$ overhead in spanner sizes from previous parallel algorithms while losing constant factors in the stretch. On unweighted graphs, this improvement comes mainly from the ability to invoke exponential start time decomposition in the spanner construction by Peleg and Schaffer algorithm [PS89]. Our extension of this routine to the weighted case relies on the probabilistic aspects of the decomposition. This leads to improvements by factors of $\frac{k}{\log k}$ in spanner size and factors of $k$ in work over the previous best [BS07]. Such routines are also directly applicable to the graph sparsification algorithm by Koutis [Kou14].

## 2.3 Hopsets

Hopsets were formalized by Cohen [Coh00] as a crucial component for parallel shortest path algorithms. The goal is to add a set of extra edges to the graph so that the $h$-hop distance in the new graph suffices for a good approximation. Let weight of a path $p$, denoted as $w(p)$, be the sum of weights of all edges on it, $w(p) = \sum_{e\in p} w(e)$. The distance between two vertices $s$ and $t$, $\text{dist}(s,t)$, is the weight of the shortest (lightest) $s$-$t$ path. Furthermore, with a set of edges $E'$, the $h$-hop distance between $s$ and $t$ in $E'$, denoted by $\text{dist}^h_{E'}(u,v)$, is defined to be the weight of the minimum weight path with at most $h$ edges between $s$ and $t$, using only edges from $E'$. If $h$ is omitted we assume $h = n$, if $E'$ is omitted we assume $E' = E$. A probabilistic version of hopsets can be described as follows:

DEFINITION 2.4. *Given a graph $G = (V, E, w)$, a $(\epsilon, h, m')$-hopset is a set of edges $E'$ such that:*

1. *$|E'| \le m'$.*
2. *Each edge $uv$ in $E'$ corresponds to a $uv$-path $p$ in $G$ such that $w(uv) = w(p)$.*
3. *For any vertices $u$ and $v$, with probability $1/2$ we have:*

$$dist^h_{E\cup E'}(u,v) \le (1+\epsilon)dist_E(u,v).$$

Given such a hopset, a result by Klein and and Subramanian [KS97] allows us to approximate the length of the path efficiently. They showed that when given an $(\epsilon, h, m')$-hopset, a shortest path can be found in $O(m\epsilon^{-1})$ work and $O(hn^\alpha\epsilon^{-1})$ depth. As a result, the main work in parallel shortest path algorithms is to efficiently compute hopsets. A summary of previous algorithms, as well as ours, is below in Figure 2.

For Cohen's algorithm, $n^\alpha$ processors are needed for parallel speedups in both the construction and query stages [1]. In our case, if $\epsilon$ is a constant, $O(\log^{3+\alpha} n)$ processors are sufficient for parallel speedups. Furthermore, once a hopset is constructed, even a constant number of processors suffices for speedups.

# 3. SPANNERS

We first describe our spanner construction. Our spanner construction in unweighted graphs has the same structure as the sequential routine by Peleg and Schaffer [PS89]: after the decomposition step, we add in single edges between adjacent clusters. We formalize this algorithm for completeness here.

**Algorithm 2** Spanner construction for unweighted graphs.

UNWEIGHTEDSPANNER$(G, \delta)$

Input: An unweighted graph $G$ and parameter $k \ge 1$.
Output: A $O(k)$-spanner of $G$.

1: Compute an exponential start time clustering with $\beta = \frac{\log n}{2k}$, let $H$ be the forest produced.
2: From each boundary vertex, add to $H$ one edge connecting to each adjacent cluster.
3: Return $H$.

The Peleg and Schaffer algorithm [PS89] relied on a bound introduced by Awerbuch [Awe85], which bounds the number of interacting clusters around a single vertex. The same bound can be obtained with exponential start time clusterings using Lemma 2.2.

COROLLARY 3.1. *In an exponential start time decomposition with parameter $\beta = \frac{\log n}{2k}$, for any vertex $v \in V$, the ball $B(v,1) = \{u \in V \mid d(u,v) \le 1\}$ intersects $O(n^{1/k})$ clusters in expectation.*

[1] A more detailed analysis leads to a tighter bound of $\Omega(\exp(\sqrt{\log n}))$

| Hop count | Size | Work | Depth | Notes |
|---|---|---|---|---|
| $O(n^{1/2})$ | $O(n)$ | $O(mn^{0.5})$ | $O(n^{0.5}\log n)$ | [KS97, SS99],exact |
| $O(\text{poly}\log n)$ | $O(n^{1+\alpha}\,\text{poly}\log n)$ | $\tilde{O}(mn^{\alpha})$ | $O(\text{poly}\log n)$ | [Coh00] |
| $(\log n)^{O((\log\log n)^2)}$ | $O\left(n^{O(\frac{1}{\log\log n})}\right)$ | $\tilde{O}\left(mn^{O(\frac{1}{\log\log n})}\right)$ | $(\log n)^{O((\log\log n)^2)}$ | [Coh00] |
| $O(n^{\frac{4+\alpha}{4+2\alpha}})$ | $O(n)$ | $O(m\log^{3+\alpha} n)$ | $O(n^{\frac{4+\alpha}{4+2\alpha}})$ | new |

**Figure 2: Performances of Hopset Constructions, omitting $\epsilon$ dependency.**

***Proof*** By Lemma 2.2, $B(v,1)$ intersects $k$ or more clusters with probability at most $(1-\exp(2\beta))^{k-1}$. Let $K$ be the number of clusters intersecting $B(v,1)$, we then have

$$\begin{aligned}\mathbf{E}[K] = \sum_{k=1}^{\infty}\mathbf{Pr}[K\geq k] &\leq \sum_{k=1}^{\infty}(1-\exp(-2\beta))^{k-1}\\ &= \frac{1}{\exp(-2\beta)}\\ &= \frac{1}{\exp(-\log n/k)}\\ &= n^{1/k}.\end{aligned}$$

■

LEMMA 3.2. *Given a connected unweighted graph and for any $k\geq 1$,* UNWEIGHTEDSPANNER *constructs a $O(k)$-spanner with high probability of expected size $O(n^{1+1/k})$. This takes $O(\log n\log^* n)$ depth with high probability and $O(m)$ work.*

***Proof*** The algorithm starts by constructing an exponential start time decomposition with parameter $\beta=\frac{\log n}{2k}$. Let $F$ be the forest obtained from the decomposition, notice that $F$ has at most $n-1$ edges. Then for each boundary vertex $v$, (i.e. $v$ is incident to an inter-cluster edge), we add one edge between $v$ and each of the adjacent clusters to our spanner. Using Corollary 3.1 and considering the ball $B(v,1)$ for each $v\in V$, we see that $O(n^{1+1/k})$ edges are added this way in expectation.

For an edge $e$ internal to a cluster, its stretch is certified by the spanning tree within the cluster, whose diameter is bounded by $O(k)$ with high probability by Lemma 2.1. If the edge $e$ has its endpoints in two different clusters, our spanner must contain some edge between these two clusters. As with high probability both of these clusters has diameter $O(k)$, the stretch of $e$ is agin bounded by $O(k)$ with high probability. The depth and work bounds also follow from Lemma 2.1. ■

This routine can be extended to the weighted setting by bucketing the edges by powers of 2. Given $G=(V,E,w)$ where $U=(\max_e w(e))/(\min_e w(e))$ is the ratio between maximum and minimum edge weights, we bucket the edges as

$$E_i=\{e\in E\mid w(e)\in[2^{i-1},2^i)\}.$$

This allows us to run the unweighted algorithm on essentially disjoint sets of edges, but leads to an overhead of $\log U$ in the total size. We reduce this overhead to $\log\log n$ using an approach introduced in [CMP+14] that's closely related to the AKPW low-stretch spanning tree algorithm [AKPW95]. We build spanners on these buckets in order, but contract the low-diameter components with smaller weights. Lemma 3.2 then allows us to bound the expected rate at which vertices are contracted, and in turn the size of the spanner.

Our contraction scheme is significantly simpler than previous ones because we will be able to ensure that edge weights in different levels differ by factors of $\text{poly}\,k$, where $k$ is the stretch factor. To this end, we first break up the input graph into $O(\log k)$ graphs where edge lengths are well separated. We define $G_j$ to be the graph with vertex set $V$ and edge set $\cup_{i\geq 0}E_{j+i\cdot c\lg k}$, where $c$ is an appropriate constant[2]. It is clear that the union of $O(\log k)$ such $G_j$s form the whole graph, and they all have $O(\log U)$ buckets of edges, where weights differ by at least $O(k^c)$ between different buckets. Thus if we can find a $O(n^{1+1/k})$-sized spanner for each of $G_j$, we obtain a $O(n^{1+1/k}\log k)$-sized spanner for $G$.

For each $G_j$, we use the fact that the weights are well-separated to form hierarchical contraction schemes. Pseudocode of this algorithm is given in Algorithm 3.

**Algorithm 3** Spanner Construction on graphs with well separated edge weights.

WELLSEPARATEDSPANNER($G$)
Input: A weighted graph $G$ with well separated edge weight buckets as described above.
Output: A $O(n)$-sized spanner for $G$.

1: Relabel the edge buckets to be $A_1,A_2,\ldots,A_s$ in increasing order of weights, such that edges in $A_i$ have weights in $[w_i,2w_i)$ and $w_{i+1}/w_i\geq O(k)$.
2: Initialize $H_0=\emptyset$ and $S=\emptyset$.
3: **for** $i=1$ to $s$ **do**
4: Let $\Gamma_i=G[A_i]/H_{i-1}$ with uniform edge weights.
5: Perform ESTCLUSTER with $\beta=\frac{\log n}{2k}$ on $\Gamma_i$
6: Let $F$ be the forest produced in the previous step.
7: $S=S\cup F$ and $H_i=H_{i-1}\cup F$.
8: For each boundary vertex, add one edge connecting each of the adjacent clusters to $S$.
9: **return** $S$.

THEOREM 3.3. *Given a weighted graph $G$ with $n$ vertices, $m$ edges and for any $k\geq 1$, we can compute with hight probability a $O(k)$-spanner for $G$ of expected size $O(n^{1+1/k}\log k)$, using $O(\log n\log^* n\log U)$ depth and $O(m)$ work.*

***Proof*** As discussed above, we break $G$ into $O(\log k)$ edge-disjoint graphs in which edge weights are well separated. We run WELLSEPARATEDSPANNER on each of these graphs in parallel, each iterations of the loop performs an exponential start time decomposition on disjoint sets of edges, thus the

[2]The constant $c$ should be chosen to achieve the desired succes probability from Lemma 2.1. We will hide $c$ inside big-O notations from now on.

overall work is $O(m)$. As there are $O(\log U)$ iterations, the overall depth is $O(\log n \log^* n \log U)$ with high probability.

Now we show that WellSeparatedSpanner produces a spanner for each of these graphs. In each iteration of the for-loop, the unweighted algorithm is run on $\Gamma_i = G[A_i]/H_{i-1}$. This produces an unweighted spanner for edges in $A_i$ by Lemma 3.2. Since the edge weights differ by at least $O(k)$ between levels, using Lemma 2.1 and induction on the loop index we see that vertices in the quotient graph $\Gamma_i$ corresponds to pieces of diameter at most $w_i$ in the spanner constructed so far, with high probability. Therefore the stretch bound for edges from $A_i$ in $\Gamma_i$ gets worse by at most a factor of 2 when translated in $G$.

We finish by bounding the size of our spanner. Using an argument similar to the proof of Lemma 2.1, we notice that any non-singleton vertex in one of $\Gamma_i$ has probability at least $1/n^{1/k}$ of being contracted away. Thus in expectation each vertex participates in $O(n^{1/k})$ level of WellSeparatedSpanner, and in each level it contributes $O(n^{1/k})$ inter-cluster edges and $O(1)$ forest edges in expectation. Thus WellSeparatedSpanner produces a spanner of size $O(n^{1+2/k})$, where the exponent $1 + 2/k$ can be reduced to $1 + 1/k$ if we back down on the stretch bound by a factor of 2. Since the graph is decomposed into $O(\log k)$ well separated graphs, this gives us the $O(n^{1+1/k} \log k)$ overall bound on the expected size of our spanner. ■

# 4. HOPSETS IN UNWEIGHTED GRAPHS

Our hop-set construction is based on recursive application of the exponential start time clustering from Section 2.1. We will designate some of the clusters produced, specifically the larger ones, as special. Since each vertex belongs to at most one cluster, there cannot be too many large clusters. As a result we can afford to compute distances from their centers to all other vertices, and keep a subset of them as hopset edges in the graph. There are two kinds of edges that we keep:

1. *star edges* between the center of a large cluster and all vertices in that cluster.
2. *clique edges* between the center of a large cluster and the centers of all other large clusters.

In other words, in building the hopset we put a star on top of each large cluster and connect their centers into a clique. Then if our optimal $s$-$t$ path $p^*$ encounters two or more of these large clusters, we can jump from the first to the last by going through their centers. One possible interaction between the decomposition scheme and a path $p^*$ in one level of the algorithm is shown in Figure 3.

This allows us to replace what hopefully is a large part of $p^*$ with only three edges: two star edges and one clique edge. However this replacement may increase the length of the path by the diameter of the large clusters. But as this distortion can only happen once, it is acceptable as long as the diameter of the clusters are less than $\epsilon w(p^*)$. Our algorithm then recursively builds hopsets on the small clusters. The probabilistic guarantees of an edge being cut gives that $p^*$ does not interact with too many such clusters. So once again a reasonable distortion within these clusters can be incurred.

Formally, two parameters are crucial to our algorithm: the parameter $\beta$ with which the decomposition routine is run, and the threshold $\rho$ by which a cluster is deemed large. The algorithm then has the following main steps:

1. Compute a exponential start time clustering with parameter $\beta$
2. Identify clusters with more than $n/\rho$ vertices as large clusters.
3. Construct star and clique edges from the centers of each large cluster.
4. Recurse on the small clusters.

Our choice of $\beta$ at each level of the recursion is constrained by the additive distortion that we can incur. Consider a cluster obtained at the $i^{\text{th}}$ level of the decomposition ran with $\beta_i$. Since the path has length $d$ and each edge is cut with probability $\beta_i$, the path is expected to be broken into $\beta_i d$ pieces. Therefore on average, the length of each piece in a cluster is about $\beta_i^{-1}$. The diameter of a cluster in the next level on the other hand can be bounded by $k\beta_{i+1}^{-1} \log n$, where the constant $k \geq 1$ can be chosen with desired success probability using Lemma 2.1. Therefore, we need to set $\beta_{i+1}$ so that:

$$k\beta_{i+1}^{-1} \log n \leq \epsilon \beta_i^{-1}$$
$$\beta_{i+1} \geq \left(k\epsilon^{-1} \log n\right) \beta_i.$$

In other words, the $\beta$s need to increase from one level to the next by a factor of $k\epsilon^{-1} \log n$ where $\epsilon < 1$ the distortion parameter. This means that the path $p^*$ is cut with granularity that increases by a factor of $O(\epsilon^{-1} \log n)$ each time. Note that the number of edges cut in all level of the recursion serves as a rough estimate to the number of hops in our shortcut path. Therefore, a different termination condition is required to ensure that the path is not completely shattered by the decomposition scheme. As we only recurse on small clusters, if we require their size to decrease at a much faster rate than the increase in $\beta$, our recursion will terminate with most pieces of the path within large clusters. To do this, we introduce a parameter $\rho$ to control this rate of decrease. Given a cluster with $n$ vertices, we designate a cluster $X_i$ to be small if $|X_i| \leq n/\rho$. As our goal is a faster rate of decrease, we will set $\rho = \left(k\epsilon^{-1} \log n\right)^{\delta}$ for some $\delta > 1$.

Pseudocode of our hopset construction algorithm is given in Algorithm 4. Two additional parameters are needed to control the first and last level of the recursion: $\beta = \beta_0$ is the decomposition parameter on the top level, and $n_{final}$ is the base case size at which the recursion stops.

We start with the following simple claim about the $\beta$ parameters in the recursion.

Claim 4.1. *If the top level of the recursion is called with $\beta = \beta_0$ as the input parameter. then the parameter $\beta$ in $i^{th}$ a level, denoted $\beta_i$, is given by $\beta_i = \left(k\epsilon^{-1} \log n\right)^i \beta_0$.*

We now describe how hopsets are used to speed up the parallel BFS. We prove the lemma in the generalized weighted setting as it will become useful in Section 5.

Lemma 4.2. *Given a weighted graph $G = (V, E, w)$ with $|V| = n$ and $|E| = m$, let $E'$ be the set of edges added by running* HopSet$(V, E, \beta_0)$. *Then for any $u, v \in V$, we have with probability at least $1/2$:*

$$dist^h_{E \cup E'}(u, v) \leq dist_E(u, v) + O(\epsilon \log_\rho n \cdot dist_E(u, v))$$

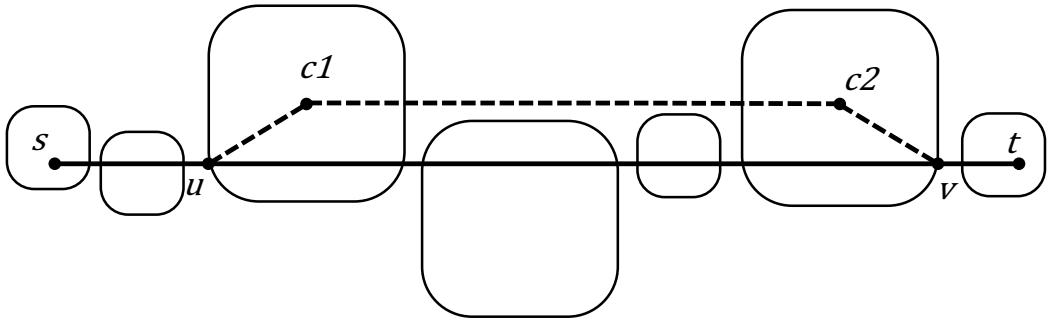


**Figure 3: Interaction of an $s-t$ path with the decomposition scheme. Shortcut edges connecting the centers of large clusters allow us to 'jump' from the first vertex of $p^*$ in a large cluster ($u$), to the last vertex of $p^*$ in a large cluster ($v$). The edges $uc_1$, $c_2v$ are star edges, while $c_1c_2$ is a clique edge.**

**Algorithm 4** Hop Set Construction on Unweighted Graphs

HopSet$(V, E, \beta)$.

Input: Undirected, unweighted graph $G = (V, E)$ and decomposition parameter $\beta$.

Output: The input graph $G$ augmented with a set of weighted edge $E^*$.

1: If $|V| \leq n_{final}$, exit.
2: Compute a exponential start time clustering of $G$ with parameter $\beta$, $\mathcal{X}$.
3: **if** this is first call **then**
4: For each cluster $X$ and *in parallel*, recursively call HopSet$(X, E(X), (k\epsilon^{-1}\log n)\beta)$.
5: **else**
6: Let $\mathcal{X}_b = \{X \in \mathcal{X} \ : \ |X| \geq |V|/\rho\}$ be the set of large clusters.
7: Let $\mathcal{X}_s = \{X \in \mathcal{X} \ : \ |X| < |V|/\rho\}$ be the set of small clusters.
8: For each large cluster $X \in \mathcal{X}_b$ with center $c$ and $v \in X$, add a star edge $(v, c)$ with weight $\text{dist}(v, c)$
9: For all pairs of large clusters $X_1, X_2 \in \mathcal{X}_b$ with centers $c_1$, $c_2$ respectively, add a clique edge $(c_1, c_2)$ with weight $\text{dist}(c_1, c_2)$.
10: For each $X \in \mathcal{X}_s$, recursively call HopSet$(X, E(X), (k\epsilon^{-1}\log n)\beta)$ *in parallel.*

*where* $h = n_{final}^{1-1/\delta} n^{1/\delta} \beta_0 \, dist_E(u, v)$.

***Proof*** Let $p$ be any shortest path with endpoints $u$ and $v$, we show how to transform it into a path $p'$ satisfying the above requirements using edges in $E'$. In each level of the algorithm, the clustering routine breaks $p$ up into smaller pieces by cutting some edges of $p$. Consider an input subgraph in the recursion that intersects the path $p$ from vertex $x$ to vertex $y$. The decomposition partitions this intersection into a number of segments, each contained in a cluster. Starting from $x$, we can identify the first segment that is contained in a large cluster, whose start point is denoted by $x'$, and similarly we can find the last segment contained in a large cluster with its end point denoted by $y'$. We drop all edges on $p$ between $x'$ and $y'$ and reconnect them using three edges $(x', c(x'))$, $(c(x'), c(y'))$ and $(c(y'), y')$. We will refer to this procedure as shortcutting. We then recursively build the shortcuts on each segment before $u'$ and after $v'$. Note that these segments are all contained in small clusters, thus they are also recursed on during the hopset construction. We stop at the base case of our hopset algorithm.

We first analyze the number of edges in the final path $p'$, obtained by replacing some portion of $p$ with shortcut edges. The path $p'$ consists of edges cut by the decomposition, shortcut edges that we introduced, and segments that are contained in base case pieces. It suffices to bound the number of cut edges, as the segments in $p'$ separated by the cut edges have size at most the size of the base case. Recall from Lemma 2.2 that any edge of weight $w(e)$ has probability $\beta w(e)$ of being cut in the clustering. Thus, the expected number of cut edges can be bounded by

$$\sum_{e \in p} \left( \sum_i \beta_i \right) w(e) = \left( \sum_i \beta_i \right) w(p).$$

Since $\beta_i$s are geometrically increasing, we can use the approximation $\sum_i \beta_i \approx \beta_l$, where $l = \log_\rho n$ is the depth of recursion. Recalling that $\rho = (k\epsilon^{-1}\log n)^\delta$:

$$\begin{aligned}
\beta_l d &= \left(k\epsilon^{-1}\log n\right)^{\log_\rho\left(\frac{n}{n_{final}}\right)} \beta_0 d \\
&= \left(\rho^{1/\delta}\right)^{\log_\rho\left(\frac{n}{n_{final}}\right)} \beta_0 d \\
&= \left(\frac{n}{n_{final}}\right)^{1/\delta} \beta_0 d.
\end{aligned}$$

As the recursion terminates when clusters have fewer than $n_{final}$ vertices, each path in such a cluster can have at most $n_{final}$ hops. Multiplying in this factor gives $n^{1/\delta} n_{final}^{1-1/\delta} \beta_0 d$.

Next we analyze the distortion introduced by $p'$ compared to the original path $p$. Shortcutting in level $i$ introduces an additive distortion of at most $4c\beta_i^{-1}\log n$. The expected number of shortcut made in level $i$, in other words the expected number of cluster in $(i-1)^{\text{th}}$ level intersecting the path $p$, is bounded by $\beta_{i-1}d$. Thus the amount of additive distortion introduced in level $i$ is at most

$$(\beta_{i-1}d) \cdot \frac{4c\log n}{\beta_i} = O(\epsilon d).$$

This gives an overall additive distortion of $O(\epsilon d \log_\rho n)$. ■

Lemma 4.3. *If* HopSet *is run on a graph $G$ with $n$ vertices, it adds at most $n$ star edges and $O\left((n/n_{final})\log^{2\delta} n\right)$ clique edges to $G$.*

***Proof*** As we do not recurse on large clusters, each vertex is part of a large cluster at most once. As a result, we add at most $n$ edges as star edges in Line 8 of HopSet.

To bound the number of clique edges, we claim that the worst case is when we always generate small clusters, except in the level above the base cases, where all the clusters are large. Suppose an adversary trying to maximize the number of clique edges decides which clusters are large. Since we do not recurse on large clusters, if on any level above the base case we have a large cluster, the adversary can always replace it with a small cluster, losing at most $\rho$ clique edges doing so (since there are at most $\rho$ large clusters), and gain $\rho^2$ edges in the next level by making the algorithm recurse on that cluster. Since the base case clusters have size at most $n_{final}$, there are at most $n/n_{final}$ clusters in the level above, where each cluster adding at most $\rho^2$ edges. Therefore at most $(n/n_{final})\rho^2 = (n/n_{final})(\log n/\epsilon)^{2\delta}$ edges are added in total. ■

Theorem 4.4. *Given constants $\delta > 1$ and $\gamma_1 < \gamma_2 < 1$, we can construct a $(\epsilon \log n, h, O(n))$-hopset on a graph with $n$ vertices and $m$ edges in $O(n^{\gamma_2} \log^2 n \log^* n)$ depth and $O(m \log^{1+\delta} n\epsilon^{-\delta})$ work, where $h = n^{1+1/\delta+\gamma_1(1-1/\delta)-\gamma_2}$.*

***Proof*** We claim that the theorem statement can be obtained by setting $\beta_0 = n^{-\gamma_2}$ and $n_{final} = n^{\gamma_1}$. The correctness of the constructed hopset follows directly from Lemma 4.2, Lemma 4.3, and the fact that any path in an unweighted graph has weight at most $n$. Specifically, for any vertices $u$ and $v$ with $\text{dist}(u, v) = d$, the expected hop-count is:

$$\begin{aligned} n^{1/\delta} n_{final}^{1-1/\delta} \beta_0 d &\leq n^{1/\delta} n^{\gamma_1(1-1/\delta_2)} n^{-\gamma_2} n \\ &= n^{1+1/\delta+\gamma_1(1-1/\delta)-\gamma_2} \end{aligned}$$

and the expected distortion is $O(\epsilon \log n \cdot d)$. By Markov's inequality, the probability of both of these exceeding four times their expected value is at most $1/2$, and the result can be obtained by adjusting the constants.

So we focus on bounding the depth and work. As the size of each cluster decreases by a factor of $\rho$ from one level to the next, the number of recursion levels is bounded by $\log_\rho(n/n_{final})$. As $n/n_{final}$ is polynomial in $n$ with our choice of parameters, we will treat this term as $\log n$.

The algorithm starts by calling $\text{HopSet}(V, E, n^{-\gamma_2})$. Since the recursive calls are done in parallel, it suffices to bound the time spent in a single call on each level. Lemma 2.1 gives that the clustering takes $O(\beta^{-1} \log n \log^* n)$ depth and linear work. Since the value of $\beta$ only increases in subsequent levels, all decompositions in each level of the recursion can be computed in $O(n^{\gamma_2} \log n \log^* n)$ depth and $O(m)$ work. This gives a total of $O(n^{\gamma_2} \log^2 n \log^* n)$ depth and $O(m \log n)$ work from Line 2. In addition, Line 8 can be easily incorporated into the decomposition routine at no extra cost.

To compute the all-pair shortest distances between the centers of the large clusters (Line 9), we perform the parallel BFS by [UY91] from each of the centers. By Lemma 2.1, the diameter of the input graphs to recursive calls after the top level is bounded by $O(n^{\gamma_2} \log n)$. Therefore the parallel BFS only need to be ran for $O(n^{\gamma_2} \log n)$ levels. This gives a total depth of $O(n^{\gamma_2} \log n \log^* n)$ and work of $O(\rho m)$ per level. Summing over $O(\log n)$ levels of recursion gives $O(n^{\gamma_2} \log^2 n \log^* n)$ depth and $O(\rho m \log n) = O(\epsilon^{-\delta} m \log^{1+\delta} n)$ work. ■

The unweighted version of Theorem 1.2 then follows from Theorem 4.4 by setting $\delta = 1 + \alpha$, and solving $h = n^{\gamma_2}$ to balance the depth for hopset construction and the depth for finding approximate distances using hopsets [KS97]. For a concrete example of setting these parameters, $\delta = 1.1$, $\epsilon = \frac{\epsilon'}{\log n}$, $\gamma_2 = 0.96$, and setting $\gamma_1$ to some small constant leads to the following bound.

Corollary 4.5. *For any constant $\epsilon' > 0$, there exists an algorithm for finding $(1 + \epsilon')$-approximation to unweighted $s - t$ shortest path that runs in $O(n^{0.96} \log^2 n \log^* n)$ depth and $O(m \log^{3.2} n)$ work.*

# 5. HOPSETS IN WEIGHTED GRAPHS

In this section we show how to construct hopsets in weighted graphs with positive edge weights. We will assume that the ratio between the heaviest and the lightest edge weights is $O(n^3)$. This is due to a reduction similar to the one by Klein and Subramanian [KS97]. In that work, they partition the edges into categories with weights between powers of 2, and show that only considering edges from $O(\log n)$ consecutive categories suffices for approximate shortest path computation. This scheme can be modified by choosing categories with powers of $n$, and then considering a constant number of consecutive categories suffices for good approximations. This result is summarized in the following lemma, we refer the readers to the full version of this paper [MPVX14] for the full proof.

Lemma 5.1. *Given a weighted graph $G = (V, E, w)$, we can efficiently construct a collection of graphs with $O(|V|)$ vertices and $O(|E|)$ edges in total, such that the edge weights in any one of these graphs are within $O(n^3)$ of each other. Furthermore, given a shortest path query, we can map it to a query on one of the graphs whose answer is a $(1 - \epsilon)$-approximation for the original query.*

Recall that the parallel BFS of [UY91] conducts the search level by level, and divides the work of each level between the processors. So a simple adaptation of parallel BFS to weighted graphs can lead to depth linear in path lengths, which can potentially be big even though the number of edge hops is small. To alleviate this we borrow a rounding technique from [KS97]. The main idea is to round up small edge weights and pay a small amount of distortion, so that the search advances much faster.

Suppose we are interested in a path $p$ with at most $k$ edges whose weight is between $d$ and $cd$. We can perturb the weight of each edge additively by $\frac{\zeta d}{k}$ without distorting the final weight by more than $\zeta d$. This value serves as the "granularity" of our rounding, which we denote using $\hat{w}$:

$$\hat{w} = \frac{\zeta d}{k}$$

for some $0 < \zeta < 1$ and round the edge weights $w(e)$ to $\tilde{w}(e)$

$$\tilde{w}(e) = \left\lceil \frac{w(e)}{\hat{w}} \right\rceil.$$

Notice that this rounds edge weights to multiples of $\hat{w}$. The properties we need from this rounding scheme is summarized in the following lemma.

Lemma 5.2 (Klein and Subramanian [KS97]). *Given a weighted graph and a number $d$. Under the above rounding*

*scheme, any shortest path $p$ with size at most $k$ and weight $d \leq w(p) \leq cd$ for some $c$ in the original graph now has weight $\tilde{w}(p) \leq \lceil ck/\zeta \rceil$ and $\hat{w} \cdot \tilde{w}(p) \leq (1+\zeta)w(p)$.*

Thus we only need to run weighted parallel BFS for $O(ck\zeta^{-1})$ levels to recover $p$, giving a depth of $O(ck\zeta^{-1} \log n)$. Therefore, if we set $c = n^{\eta}$ for some $\eta < 1$, and since the edge weights are within $O(n^3)$ of each other, we can just try building hopsets using $O(3/\eta)$ estimates, incurring a factor of $O(3/\eta)$ in the work. As one of the values tried satisfies $d \leq w(p) \leq cd$, Lemma 5.2 gives that if $\zeta$ is set to $\epsilon/2$, an $(1+\epsilon/2)$-approximation of the shortest path in the rounded graph is in turn an $(1+\epsilon)$-approximation to the shortest path in the original graph. Therefore, from this point on we will focus on finding an $(1+\epsilon)$-approximation of the shortest path in the rounded graph with weights $\tilde{w}(e)$. In particular, we have that all edge weights are positive integers, and the shortest path between $s$ and $t$ has weight $O(n^{1+\eta}/\zeta) = O(n^{1+\eta}/\epsilon)$.

THEOREM 5.3. *For any constants $\delta > 1$ and $\gamma_1 < \gamma_2 < 1$, we can construct a $(\epsilon \log n, h, O(n))$-hopset on a graph with $n$ vertices and $m$ edges in expected $O((n/\epsilon)^{\gamma_2} \log^2 n \log^* n)$ depth and $O(m \log^{1+\delta} n\epsilon^{-\delta})$ work, where $h = n^{1+1/\delta+\eta+\gamma_1(1-1/\delta)-\gamma_2}/\epsilon^{1-\gamma_2}$.*

***Proof*** Since the edge weights are within a polynomial of each other, we can build $O(1/\eta)$ hopsets in parallel for all values of $d$ being powers of $n^{\eta}$. For any pair of vertices $s$ and $t$, one of the value tried will satisfy $d \leq \text{dist}(s,t) \leq n^{\eta}d$. Given such an estimate, we first perform the rounding described above, then we run Algorithm 4 with $\beta = (n/\epsilon)^{-\gamma_2}$ and $n_{final} = n^{\gamma_1}$. The exponential start time clustering in Line 2 takes place in the weighted setting, and Line 9 becomes a weighted parallel BFS. The correctness of the hopset constructed follows from Lemma 4.2, Lemma 4.3, and the fact that $\text{dist}(s,t) = O(n^{1+\eta}/\epsilon)$ by the rounding. Specifically, the expected hop count is

$$n^{1/\delta} n_{final}^{1-1/\delta} \beta d \leq n^{1/\delta} n^{\gamma_1(1-1/\delta)} \left(\frac{n}{\epsilon}\right)^{-\gamma_2} \frac{n^{1+\eta}}{\epsilon}$$
$$= n^{1+1/\delta+\eta+\gamma_1(1-1/\delta)-\gamma_2}/\epsilon^{1-\gamma_2}$$

and the expected distortion is $O(\epsilon d)$. By Markov's inequality, the probability of both of these exceeding four times their expected values is at most $1/2$.

The number of recursion levels is still bounded by $\log_{\rho} n$. Since the $\beta$s only increase, according to Lemma 2.1 we spend $O((n/\epsilon)^{\gamma_2} \log n \log^* n)$ depth in each level of the recursion and $O((n/\epsilon)^{\gamma_2} \log^2 n \log^* n)$ overall in Line 2. Since our decomposition is laminar, we spend $O(m)$ work in each level and $O(m \log n)$ overall in Line 2. Again, Line 8 can be incorporated into the decomposition with no extra cost.

Since the diameter of the pieces below the top level is bounded by $\beta^{-1} \log n = (n/\epsilon)^{\gamma_2} \log n$ and the minimum edge weight is one, Line 9 can be implemented by weighted parallel BFS in depth $O((n/\epsilon)^{\gamma_2} \log n \log^* n)$ in one level and $O((n/\epsilon)^{\gamma_2} \log^2 n \log^* n)$ in total. The work done by the weighted parallel BFS is $O(\rho m)$ per level and $O(\rho m \log n) = O(m \log^{1+\delta} n\epsilon^{-\delta})$ in total. ■

Theorem 1.2 then follows from Theorem 5.3 by adjusting the various parameters. Again, to give a concrete example, we can set $\delta = 1.1$, $\epsilon = \epsilon'/(\log n)$, $\gamma_2 = 0.96$, and set $\gamma_1$ and $\zeta$ to some small constants to obtain the following corollary

COROLLARY 5.4. *For any constant error factor $\epsilon'$, there exists an algorithm for finding $(1+\epsilon')$-approximation to weighted $s$-$t$ shortest path that runs in $O(n^{0.96} \log^2 n \log^* n)$ depth and $O(m \log^{3.2} n)$ work in a graph with polynomial edge weight ratio.*

Notice that with our current scheme it is not possible to push the depth under $\tilde{O}(\sqrt{n})$ as the hop count becomes the bottle neck. A modification that allows us to obtain a depth of $\tilde{O}(n^{\alpha})$ for arbitrary $\alpha > 0$ at the expense of incurring more work can be found in the full version of this paper [MPVX14].

## Acknowledgements

Miller and Xu are supported by the National Science Foundation under grant number CCF-1018463 and CCF-1065106. Peng was partially supported by a Microsoft Research Ph.D. Fellowship.

We thank Timothy Chu and Yiannis Koutis for helpful discussions.

## 6. REFERENCES

[ADD+93] Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete Comput. Geom.*, 9(1):81–100, January 1993. 2.2

[AKPW95] N. Alon, R. Karp, D. Peleg, and D. West. A graph-theoretic game and its application to the $k$-server problem. *SIAM J. Comput.*, 24(1):78–100, 1995. 1, 3

[Awe85] Baruch Awerbuch. Complexity of network synchronization. *J. Assoc. Comput. Mach.*, 32(4):804–823, 1985. 1, 3

[Bar96] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 184–193. IEEE, 1996. 1

[BGK+14] Guy E. Blelloch, Anupam Gupta, Ioannis Koutis, Gary L. Miller, Richard Peng, and Kanat Tangwongsan. Nearly-linear work parallel SDD solvers, low-diameter decomposition, and low-stretch subgraphs. *Theory Comput. Syst.*, 55(3):521–554, 2014. 1

[BKMP05] Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. New constructions of $(\alpha, \beta)$-spanners and purely additive spanners. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 672–681, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics. 2.2

[BKMP10] Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. Additive spanners and $(\alpha, \beta)$-spanners. *ACM Transactions on Algorithms (TALG)*, 7(1):5, 2010. 2.2

[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, July 2007. 2.2, 2.2

[CMP+14] Michael B. Cohen, Gary L. Miller, Jakub W. Pachocki, Richard Peng, and Shen Chen Xu. Stretching stretch. *CoRR*, abs/1401.2454, 2014. 1, 3

[Coh98] Edith Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM J. Comput.*, 28(1):210–236, 1998. 1

[Coh00] Edith Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *J. ACM*, 47(1):132–166, 2000. 1, 2.3, 2.3

[DHZ00] Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM J. Comput.*, 29(5):1740–1759, March 2000. 2.2

[GMV91] Joseph Gil, Yossi Matias, and Uzi Vishkin. Towards a theory of nearly constant time parallel algorithms. In *FOCS*, pages 698–710. IEEE Computer Society, 1991. A

[Kou14] Ioannis Koutis. Simple parallel and distributed algorithms for spectral graph sparsification. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '14, pages 61–66, New York, NY, USA, 2014. ACM. 2.2

[KS97] Philip N Klein and Sairam Subramanian. A randomized parallel algorithm for single-source shortest paths. *Journal of Algorithms*, 25(2):205–220, 1997. 1, 2.3, 4, 5, 5, 5.2, A

[KSV10] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 938–948, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics. 2

[MPVX14] Gary L. Miller, Richard Peng, Adrian Vladu, and Shen Chen Xu. Improved parallel algorithms for spanners and hopsets. *CoRR*, abs/1309.3545, 2014. 5, 5

[MPX13] Gary L. Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In *Proceedings of the 25th ACM symposium on Parallelism in algorithms and architectures*, SPAA '13, pages 196–203, New York, NY, USA, 2013. ACM. 1, 2.1, 2.1

[Pel00] David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000. 2.2

[Pet08] Seth Pettie. Distributed algorithms for ultrasparse spanners and linear size skeletons. In *Proceedings of the Twenty-seventh ACM Symposium on Principles of Distributed Computing*, PODC '08, pages 253–262, New York, NY, USA, 2008. ACM. 2.2

[PS89] David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989. 2.2, 2.2, 3, 3

[RTZ05] Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In *Proceedings of the 32nd International Conference on Automata, Languages and Programming*, ICALP'05, pages 261–272, Berlin, Heidelberg, 2005. Springer-Verlag. 2.2

[SDB14] Julian Shun, Laxman Dhulipala, and Guy Blelloch. A simple and practical linear-work parallel algorithm for connectivity. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '14, pages 143–153, New York, NY, USA, 2014. ACM. 1, A

[SS99] Hanmao Shi and Thomas H. Spencer. Timework tradeoffs of the single-source shortest paths problem. *J. Algorithms*, 30(1):19–32, January 1999. 2.3

[TZ05] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, Jan 2005. 1, 2.2

[UY91] J. Ullman and M. Yannakakis. High-probability parallel transitive-closure algorithms. *SIAM Journal on Computing*, 20(1):100–125, 1991. 4, 5

# APPENDIX

## A. DEFERRED PROOFS

We now show the properties of the exponential start time clustering routine from Section 2.1 in more detail.

***Proof of Lemma 2.1:***

The bound on diameter of the clusters follows from taking a union bound on the maximum value of $\delta_u$ at vertices:

$$\begin{aligned}\Pr\left[\delta_{\max} > \frac{k\log n}{\beta}\right] &\leq \sum_{v\in V}\Pr\left[\delta_v > \frac{k\log n}{\beta}\right] \\ &= n\cdot\exp\left(-\beta\cdot\frac{k\log n}{\beta}\right) \\ &= \frac{1}{n^{k-1}}.\end{aligned}$$

To compute the output of ESTCluster efficiently, we can add a super-source and connect it to vertex $u$ via an edge of length $\delta_u$, then we build a shortest path tree in increasing order of distance. The clusters then correspond to the subtrees below the super-source. We can construct this shortest path tree level by level in $O(\frac{\log n}{\beta})$ steps, taking only the integer part of the $\delta_u$s into consideration with arbitrary tie breaking in the search. Since we assume the minimum edge weight is 1, this modification in implementation can be shown to have negligible effect on the probabilistic guarantee from Lemma 2.2 (see Theorem 2 from [SDB14]). The overhead of $O(\log^* n)$ per search level comes from the overhead of CRCW PRAM [GMV91]. We remark that this factor of $\log^* n$ depends on the model of parallelism, but is standard in parallel BFS algorithms [KS97]. It is $O(1)$ in the OR CRCW PRAM model, and can be bounded by $O(\log n)$ in most models of parallelism. ■

***Proof of Lemma 2.2:***

Let $B$ be a subgraph of $G$ with center $c$ and radius $r$. From $c$'s point of view, the algorithm can be seen as a race between all the vertices to $c$: vertex $v$ starts its race at time $\delta_{\max}-\delta_v$, and arrives at $c$ at time $d(v,c)+\delta_{\max}-\delta_v$. In particular, the winner of this race will include $c$ in its cluster. For $B$ to intersect $k$ or more clusters, the first $k$ arrivals at $c$ must be

within $2r$ units of time of each other. Since $\delta_{\max}$ is a common term in everyone's arrival time, we can drop it and flip the sign to obtain a quantity $Y_v = \delta_v - d(v, x)$, for each vertex $v$. Notice that it is just an exponential random variable with some constant offset. The event we are interested in then becomes: the $k$ largest $Y_v$'s are within $2r$ of each other.

We will use the law of total probability for continuous random variables. Let $S$ vary over subsets of $V$ of size $k-1$, $u \in V \setminus S$, and $\alpha$ a fix real number. Let $E_{S,u,\alpha}$ be the event that $Y_u = \alpha$ and $Y_v \geq \alpha$ if $v \in S$ and $Y_v < \alpha$ if $v \notin S$. That is, set $S$ represents the first $k-1$ arrivals, and $u$ is the $k$th arrival at time $\alpha$. Clearly, ranging over all possible $S$, $u$ and $\alpha$ gives a paritition of the probability space. Thus it suffices to show

$$\Pr[Y_v \leq \alpha + 2r \text{ for all } v \in S \mid E_{S,u,\alpha}] \leq (1 - \exp(-2r\beta))^{k-1}$$

for any fixed $S$, $u$ and $\alpha$.

By independence of the $Y_v$s we have that

$$\begin{aligned} & \Pr[Y_v \leq \alpha + 2r \text{ for all } v \in S \mid E_{S,u,\alpha}] \\ = & \prod_{v \in S} \Pr[Y_v \leq \alpha + 2r \mid \alpha \leq Y_v]. \end{aligned}$$

For each $v \in S$,

$$\begin{aligned} & \Pr[Y_v \leq \alpha + 2r \mid \alpha \leq Y_v] \\ = & \Pr[\delta_v \leq \alpha + 2r + d(v, c) \mid \alpha + d(v, c) \leq X_v]. \end{aligned}$$

There are two cases to consider. If $\alpha + d(v, c) \leq 0$, then by the definition of the exponential distribution

$$\begin{aligned} & \Pr[\delta_v \leq \alpha + 2r + d(v, c) \mid \alpha + d(v, c) \leq X_v] \\ \leq & \Pr[\delta_v \leq 2r] \\ = & 1 - \exp(1 - 2r\beta). \end{aligned}$$

If $\alpha + d(v, c) > 0$, using the memoryless property of the exponential distribution, we have

$$\begin{aligned} & \Pr[\delta_v \leq \alpha + 2r + d(v, c) \mid \alpha + d(v, c) \leq X_v] \\ = & \Pr[\delta_v \leq 2r] \\ = & 1 - \exp(1 - 2r\beta). \end{aligned}$$

This finishes the proof. ■