Stretching Stretch

Michael B. Cohen
M.I.T.*
micohen@mit.edu

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

Jakub W. Pachocki
Carnegie Mellon University
pachocki@cs.cmu.edu

Richard Peng
M.I.T.†
rpeng@mit.edu

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

November 11, 2013

Abstract

We give a generalized definition of stretch that simplifies the efficient construction of low-stretch embeddings suitable for a variety of efficient graph algorithms. The generalization is based on discounting highly stretched edges at some exponent $p < 1$. It is based on the observation that the performance of existing frameworks can be optimized within constants for any $1/2 < p < 1$.

We show that an algorithm by [Bartal STOC'96] generates high quality embeddings in this view. Furthermore, the discount allows us to treat many classes of edges with coarser granularity, allowing for a two-pass approach for generating these embeddings. It leads to an algorithm that construct such embeddings in $\mathcal{O}(m \log \log n)$ time. This algorithm also parallelizes readily.

*Part of this work was done while at CMU

†Part of this work was done while at CMU and was supported by a Microsoft Research PhD Fellowship

1 Introduction

Over the last few years substantial progress has been made on a large class of graph theoretic optimization problems. The list includes approximate solutions to such problems as maximum flow [CKM+11, KLOS13, She13], matching [Mad13], minimum cost maximum flow [DS08], minimum energy flow [ST04, KMP11, KOSZ13, CFM+14], shortest paths in arbitrarily weighted graphs [BGK+13], and cut-based problems in graphs such as sparest cut [She13]. For each of these problems we now know substantially better asymptotically run time bounds. One commonality of all these new algorithms is that they either explicitly find a Low Stretch Spanning Tree or call an algorithm that at least at present uses LSSTs. The fastest known LSST algorithm is due to Abraham and Neiman with a run time of $\mathcal{O}(m \log n \log\log n)$. The main question we address in this paper is can we find algorithms for constructing these trees with a run time $\mathcal{O}(m)$. Unfortunately this is an open question.

Of the above problems only for the approximate minimum energy flow and its dual approximate solutions to Symmetric Diagonally Dominate systems is it the case that finding a LSST is a dominate term in the run time. But there is no reason why each of the above problems cannot be solved in $\mathcal{O}(m \log n)$ time and thus the LSST construction would dominate their run time as well.

This paper removes the LSST run time obstacle from constructing $\mathcal{O}(m)$ solvers for SDD systems. To achieve this better run time we modify the definition in two substantial ways. First, we allow Steiner nodes, thus we get a Steiner tree. Second, we discount the cost of large stretch edges.

Throughout this paper we let $G = (V, E, l)$ a graph with edge lengths $l(e)$. We let $l_G(x, y)$ be the length in G from $x \in V$ to $y \in V$. If $T = (V', E', l')$ be a spanning tree on V' we can define the stretch of an edge $e = (x, y) \in E$ with respect to T,

$$\mathbf{STR}_T(e) \stackrel{\text{def}}{=} \frac{l'_T(x, y)}{l_G(e)}.$$

In most prior work it was assumed that T was a subgraph of G and that $l'(e) = l(e)$ for $e \in E'$. Since we will not be assuming this we must insure that the edge lengths in T are not too short with respect to G, which will be made formal by a notion of embeddability.

Here we adopt a notion of embeddability, which was first used by Vaidya [Vai91] and many other people including [Gre96] to bound the support of one graph by another. Up until now we have only been considering graphs with edge lengths. These lengths can also be viewed as resistive values since resistors in series sum. The inverse of a resistive value is conductance and conductances sum in parallel. We let the **weight of an edge** e to be $w(e) = 1/l(e)$. A now standard method of proving support of one graph by another is by congestion/dilation arguments, which bounds the quality of an embedding in terms of both weights and lengths.

We say that we have a **path embedding** from a graph $H = (V_H, E_H, l_H)$ to a graph $G = (V_G, E_G, l_G)$ if we have the following three functions

1. A function $\pi : V_H \to V_G$.
2. A function from each edge $e \in E_H$ to a weighted path of G, denoted by $Path(e = xy)$ that goes from $\pi(x)$ to $\pi(y)$.
3. We let $W_{Path(e)}(e')$ denote the weight of the edge e' on path $Path(e)$. This value is zero if $e' \notin Path(e)$.

Using the definition of a path embedding we now define embeddable.

Definition 1.1 *A graph H is path embeddable or simply embeddable into a graph G, if there exists a path embedding $(\pi, Path)$ of H into G such that:*

- *for all edges $e \in E_G$, $\sum_{e' \in E_H} W_{Path(e')}(e) \leq w_G$, congestion is at most one.*
- *for all edges $e' \in E_H$, $\sum_{e \in Path(e')} \frac{1}{W_P(e)} \leq l_H(e) = \frac{1}{w_H(e)}$. The dilation is at most one.*

Note that since G has no self loops the definition precludes mapping both endpoints of an edges in H to the same point in G.

Critical to our more efficient algorithm is our relaxation of edges which have larger stretch. The above definition measures the sum of stretch. There are many ways to discount these high stretch edges. In Blelloch et al. [BGK+11] the high stretch edge were simply added to the spanning tree giving a subgraph. Here we will use ℓ_p-stretch:

$$\mathbf{STR}_T^p(e) \stackrel{\text{def}}{=} (\mathbf{STR}_T(e))^p .$$

Our main result is as follows:

Theorem 1.2 *Let $G = (V, E, w)$ be a weighted graph with n vertices and m edges, and p is any constant strictly between 0 and 1. We can construct a distribution over embeddable trees that are path-embeddable in G such that for any edge e its expected ℓ_p-stretch in a tree picked from this distribution is $\mathcal{O}(\log^p n)$.*

Furthermore, a tree from this distribution can be picked in $\mathcal{O}(m \log\log n)$ time in the RAM model.

An overview of our algorithm for generating low ℓ_p-stretch embeddable trees is in Section 2. We expand on it using existing low stretch embedding algorithms in mostly black-box manners in Sections 3. Then in Section 4 we show a two-step approach that combines bottom-up and top-down routines that gives our main result.

Although our algorithm runs in $\mathcal{O}(m \log\log n)$ time, the running time is in the RAM model, and our algorithm calls sorting. As a result, our algorithm falls just short of answering the question whether SDD linear systems can be solved to constant accuracy faster than sorting. However, this dependency on sorting is rather mild, as it is only used to approximately bucket the edge weights. In the pointer model, this can be done in $\mathcal{O}(m \log_m(\log U))$ time if all edge lengths are between 1 and U. As our analyses repeatedly use the robustness of the ℓ_p-stretch definition, there should be a lot of algorithmic flexibility for working around this bottleneck. We conjecture that these trees can be generated even faster in the RAM model, as well as in $\mathcal{O}(m \log\log n)$ time in the pointer machine model. Another question of interest is whether the embeddability requirement can be strengthened back to subgraphs to form low ℓ_p-stretch spanning trees.

1.1 Related Works

Alon et al. [AKPW95] first proposed the notion of LSST and gave a routine for consturcting such trees, known as the AKPW low stretch spanning trees. They showed that for any graph, there is a distribution over spanning trees such that the expected stretch of an edge is $\mathcal{O}(\exp(\sqrt{\log n \log\log n}))$. Subsequently, results with improved expected stretch were obtained by returning an arbitrary

tree metric instead of a spanning tree. Such metrics are only required to not shorten distances in the graph, and may also include extra vertices. Bartal gave trees with expected stretch of $\mathcal{O}(\log^2 n)$ [Bar96], and $\mathcal{O}(\log n \log\log n)$ [Bar98]. The optimal $\mathcal{O}(\log n)$ embedding is found by Fakcharoenphol et al. [FRT04], and known as the FRT trees. This guarantee can be written formally as

$$\mathop{\mathbf{E}}_{T}[\mathbf{STR}_T(e)] \leq \log n.$$

Recent applications to SDD linear system solvers has led to renewed interests in finding spanning trees with improved stretch over AKPW trees. The first LSSTs with polylog stretch were given by Elkin et al. [EEST08]. They gave an algorithm where the expected stretch of an edge is $\mathcal{O}(\log^2 n \log\log n)$, which has subsequently been improved to $\mathcal{O}(\log n \log^3 \log n)$ by Abraham et al. [ABN08] and to $\mathcal{O}(\log n \log\log n)$ by Abraham and Neiman [AN12].

Notationally our guarantee is almost identical to the expected stretch above:

$$\mathop{\mathbf{E}}_{T}[\mathbf{STR}^p_T(e)] \leq \log^p n.$$

As $p < 1$, the power mean inequality implies that our embedding is weaker than ℓ_1 ones. However, at present, such guarantees for ℓ_1 are *not known*, with the closest being the result by Abraham and Neiman [AN12], which is off by a factor of $\log\log n$.

1.2 Applications

Our definition of ℓ_p-stretch embeddable trees can be used in most frameworks that reduce the size of graphs using low-stretch spanning trees. In Section 5, we check that the larger graph with Steiner trees can lead to linear operators close to the graph Laplacian of the original graph. It allows us to use these trees in algorithms for solving linear systems in graph Laplacians, and in turn SDD linear systems. This analysis also generalizes to other norms, which means that our trees can be used in approximate flow [LS13, She13] and minimum cut [Mad10] algorithms.

Combining our algorithm with recursive preconditioning framework by Koutis et al. [KMP11] leads to an algorithm that runs solves such a system to constant accuracy in $\mathcal{O}(m \log n)$ time. With the accelerated coordinate descent algorithm by Lee and Sidford [LS13], it can be checked that this leads to a running time of $\mathcal{O}(m \log^{3/2} n)$ [Sid13]. also a crucial component is the more recent solver Such trees are also needed for the faster solver by Cohen et al. [CKP+13], which runs in about $m \log^{1/2} n$ time. Here the running time of returning a tree is crucial, as the low-stretch spanning tree algorithm by Abraham and Neiman [AN12] takes $m \log n \log\log n$ time.

One side effect of our faster construction algorithm in Section 4 is that all the partition routines rely on computing shortest paths on graphs with small diameter. The parallel version of the partition routine from [MPX13] immediately allows the parallelization of our algorithm. When combined with the parallelization of the recursive preconditioning framework by [BGK+13], this leads to work efficient parallel solvers with depth of about $m^{1/3}$. In this setting, it is more useful to generate ultra-sparsifiers, which are trees plus a sub-linear number of edges that approximate the original graph. It can be checked that our algorithm can be modified to generate these in parallel as well. However, such parallelizations should also incorporate aspects of the recent polylog depth, nearly-linear work parallel solver by Peng and Spielman [PS13]. As a result, we omit the discussion of parallelization with the hope that there is a more refined parallel algorithm that combines all

these routines.

2 Overview

In this section we overview the structure of the trees that we produce, the additional embeddability requirements, and our construction of these trees. Bartal trees introduced in [Bar96] can be viewed as trees corresponding to laminar decompositions. These decompositions give several levels of refinement of the graph. At the top level, all vertices are in a single cluster, and these clusters are then broken down in subsequent levels. A tree is formed from this laminar decomposition by choosing a vertex from each cluster on each level to be the center of that cluster. Each center at a higher level is then connected to all centers in the next level that are in its cluster, leading to a star from one level to the next. This structure leads to a definition known as hierarchically well-separated trees (HSTs), which has the additional constraint that the lengths of the edges are geometrically decreasing.

We modify this structure to ensure that the resulting tree is embeddable into the graph. If a center only has one edge leaving it, but is connected to many centers on the lower level, it would be difficult to embed a high degree star through this one edge. Instead, we form trees that connect centers of each cluster to all vertices in it. These trees allow us to bound distance traveled in clusters, and are analogous to the role of the stars in HSTs. Also, as long as their diameters are geometrically decreasing, we can obtain distance bounds similar to those obtained from HSTs.

Our modified definition of Bartal decompositions is as follows:

Definition 2.1 *Let $G = (V, E, l)$ be a connected multigraph. We say that a sequence of forests $\mathcal{B}$, where*

$$\mathcal{B} = (B_0, B_1, \ldots, B_t),$$

is a Bartal decomposition *of G with the diameter sequence*

$$d_0, d_1, \ldots, d_t$$

if all of the following conditions are satisfied:

1. *For any $i \leq t$, it holds that B_i is a forest subgraph of G. In particular, B_0 is a spanning tree of G and, B_t is an empty graph.*

2. *For any pair of vertices u, v, and $i < t$, if u and v are in the same connected component of B_{i+1}, then they are in the same connected component of B_i.*

3. *For any $i \leq t$, the diameter of any connected component of B_i does not exceed d_i.*

Additionally, for an edge $e \in B_i$, we let $w_i(e)$ be its weight in B_i (not necessarily equal to its weight $w(e)$ in G), and we let $w_i(e) = 0$ if $e \notin B_i$. Then a Bartal decomposition $\mathcal{B}$ is embeddable *if $\sum_{i=0}^{t} w_i(e) = \mathcal{O}(w(e))$ for all edges e.*

We say an edge with endpoints u and v is *cut at level i* of a Bartal decomposition $\mathcal{B}$ if u, v are in the same connected component of B_i, and $i = t$ or u and v are in distinct connected components of B_{i+1}. It is also convenient for us to define the stretch of an edge w.r.t. a Bartal decomposition.

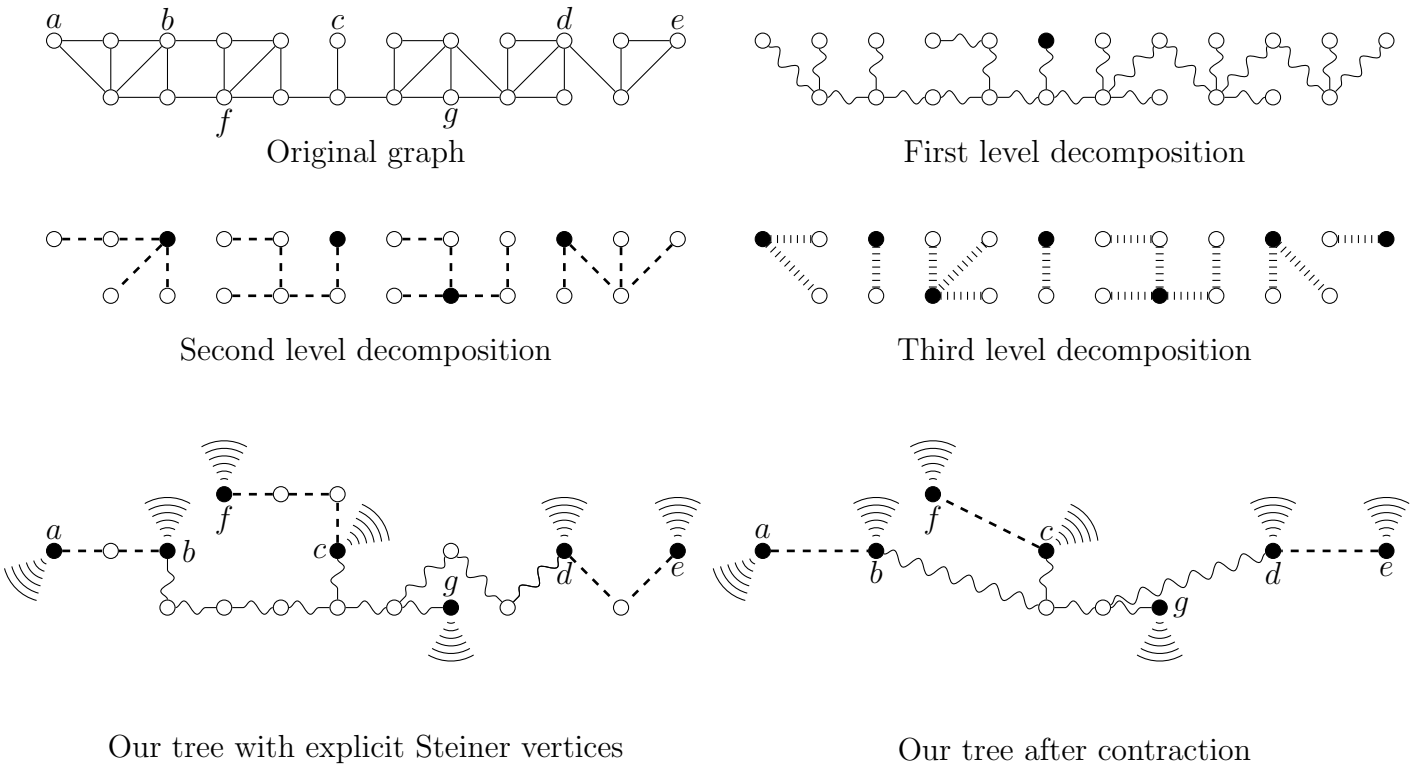

Figure 1: Bartal decomposition and the tree produced for a particular graph

Definition 2.2 *For an edge e with endpoints u, v, a Bartal decomposition $\mathcal{B}$, and a constant $p \in (0,1)$, we define the* ℓ_p-stretch *of e with respect to $\mathcal{B}$ as:*

$$\boldsymbol{STR}^p_{\mathcal{B}}(e) \stackrel{def}{=} \left(\frac{d_i}{l(e)}\right)^p,$$

where $l(e)$ is the length of e and i is the level of $\mathcal{B}$ at which e is cut.

Given such a decomposition, we can from a Steiner tree by assigning a center to each cluster on each level, and connect them using the edges in $\mathcal{B}$. At a first glance, the resulting Steiner tree may have size $\Omega(nt)$, since each B_i may have up to n edges. However, it can be reduced to a tree with at most $\mathcal{O}(n)$ vertices since only the distances between the non-Steiner nodes in G need to be preserved. In our simpler algorithm, we will first construct this tree, and then trim it into one of linear size.

We can now formalize the notion of generating a tree that is embeddable into G from a Bartal decomposition. The routine for constructing the tree is given in Figure 2, and a visual walkthrough is given in Figure 1. In this example the decomposition gives four levels of refinement to the graph, the last level where each singleton vertex forms its own cluster is omitted. On each level we have the centers of clusters filled and the spanning trees of each cluster is given. Notice that the four center vertices on the second level, b, c, d and g are connected using part of the spanning tree on the first level (wavy edges), while center vertices of the third level, a and b, are connected by spanning trees of the second level (dashed edges). Our construction of the tree relies on the following notion,

$T = \textsc{buildTree}(G, \mathcal{B})$ where $\mathcal{B}$ is a Bartal decomposition of G.

1. Designate a center vertex for each connected component of each level of $\mathcal{B}$ such that if u is a center vertex on level i, it is also a center vertex on level $i+1$.
2. For each connected component on level i.
   (a) Find all center vertices in level $i+1$ contained in this piece.
   (b) Connect these vertices using trees from B_i, adding intermediate vertices to T as Steiner vertices.
3. Return T.

Figure 2: Constructing a Steiner tree from a Bartal decomposition

analogous to the result of GreedyEliminate from [KMP11].

Definition 2.3 *We define the* contraction *of a tree T to a subset of its vertices S as the unique tree arising from repeating the following operations while possible:*

- *removal of a degree 1 vertex not in S, and*
- *contraction of a degree 2 vertex not in S.*

Fact 2.4 *For any tree T and a subset of its vertices S of size k, the contraction T_S of T to S satisfies the following properties:*

- *The distances between vertices in S are the same in T and T_S.*
- *T_S has $\mathcal{O}(k)$ vertices.*
- *T_S is embeddable into T.*

The guarantees of our tree construction can then be summarized as follows.

Lemma 2.5 *Given an embeddable Bartal decomposition $\mathcal{B}$, we can construct a tree T with $O(n)$ edges that is embeddable in G such that $\boldsymbol{STR}_T(e) = \mathcal{O}(\boldsymbol{STR}_{\mathcal{B}}(e))$ for all edges e.*

Proof Consider a tree T' returned by BuildTree given in Figure 2. Since T' consists of trees from $\mathcal{B}$, it is natural to map an edge $e \in T'$ to the edge in G from which e came. Since we are mapping edges of T' to edges of G, the dilation requirement is satisfied. Furthermore we have $\sum_i w_i(e) = \mathcal{O}(w(e))$ because $\mathcal{B}$ is embeddable, thus the congestion requirement is also satisfied. As we only need to preserve the distances between the centers and there are n centers total, together with Fact 2.4 we can establish the existence of an $\mathcal{O}(n)$ sized tree T that is embeddable into G.

To obtain the bound on stretch, consider an edge $e = uv$ and the path between them that moves upwards in the tree T'. At level i, the distance traveled is bounded by d_i, and since d_is are geometrically decreasing, it is within a constant factor of the bound of the level where e is cut.

$\mathcal{B} = \text{EMBEDDABLEDECOMPOSE}(G, p, q, \text{DECOMPOSE}_q)$, where G is a graph, p, q are exponents, and DECOMPOSE is a routine that generates a decomposition with low ℓ_q-stretch.

1. Create graph G' with edge lengths $l'(e) = l(e)^{\frac{p}{q}}$.
2. Set $\mathcal{B}' := \text{DECOMPOSE}_q(G')$.
3. Create decomposition $\mathcal{B}$ by scaling level i of $\mathcal{B}'$ by $\left(\frac{\log n}{d'_i}\right)^{\frac{q-p}{p}}$.
4. Return $\mathcal{B}$.

Figure 3: Using a generic Decomposition Routine to Generate an Embeddable Decomposition

Since T preserves distances between the centers, we have $\mathbf{STR}_T(e) = \mathbf{STR}_{T'}(e) = \mathcal{O}(\mathbf{STR}_{\mathcal{B}}(e))$ for all edges $e \in G$. ■

We should point out that this is a purely existential proof that establishes the sufficency for finding embeddable Bartal decompositions. The notion of embeddability is also robust with respect to the ℓ_p-stretch of Bartal decompositions. If we have a routine that finds Bartal decompositions with low ℓ_q-stretch, we can use it to generate embeddable Bartal decompositions with low ℓ_p-stretch for any $p < q$. Pseudocode of this routine is given in Figure 3. The algorithm calls a subroutine DECOMPOSE which we construct in Section 4, it generates a Bartal decomposition with good ℓ_q-stretch. The guarantees of the algorithm is summarized by the following theorem.

Theorem 2.6 *Let $G = (V, E, l)$ be a connected multigraph and p, q be constants such that $0 < p < q < 1$. Let $G' := (V, E, l')$, where*

$$l'(e) := l(e)^{\frac{p}{q}}$$

Given any Bartal decomposition $\mathcal{B}'$ of G', we can reweigh each level of it by the same factor to obtain a decomposition $\mathcal{B}$ that is is embeddable into G and for every edge $e \in E$ it holds that

$$\boldsymbol{STR}^p_{\mathcal{B}}(e) = \mathcal{O}\left(\log^{p-q} n \cdot \boldsymbol{STR}^q_{\mathcal{B}'}(e)\right).$$

This would imply if we can guarantee that $\mathbf{E}[\mathbf{STR}^q_{\mathcal{B}'}(e)] = O(\log^q n)$ for every $e \in E$, this would imply that $\mathbf{E}[\mathbf{STR}^p_{\mathcal{B}}(e)] = O(\log^p n)$ for every $e \in E$ and embeddability comes for free. Therefore it suffices to show the existence of Bartal decompositions where each edge is expected to have small ℓ_p-stretch. In Section 3, we give a proof for Theorem 2.6 and we show that Bartal's algorithm [Bar96] generates a random decomposition such that the expected ℓ_p-stretch of an edge is small:

Lemma 2.7 *Let $G = (V, E, w)$ be a weighted graph with n vertices and m edges with weights $w : E \to [1, \Delta]$ and any constant $p \in (0, 1)$. We can construct a distribution over Bartal decompositions such that for any edge e, its expected ℓ_p-stretch in a decomposition picked from this distribution is $\mathcal{O}(\log^p n)$.*

Furthermore, this distribution can be sampled in $\mathcal{O}(m \log(n\Delta) \log n)$ time in the RAM model.

This establishes the existence of good Bartal decompositions, and a relatively efficient algorithm for computing them when the edge weights are in a small range. However, we omit some crucial components to this algorithm, leading to a $\log \Delta$ dependency instead of the $\log n$ terms in [Bar96] as well as in other embedding algorithms. This omission is intentional, as examining how to remove it naturally leads to the faster $O(m \log \log n)$ time algorithm.

Bartal's algorithm is a top-down decomposition process where each cluster at level i is partitioned to form clusters at level $i+1$. This partition routine can be viewed as a shortest path computation, and its running time depends the weights of the edges given to it. Using the modifications given in [KMP11], this leads to a running time of $O(m \log \log \Delta)$ per partition call. The geometric decrease in diameter means that the number of levels is bounded by $O(\log \Delta)$, giving the overall running time.

To improve the performance of this algorithm, Bartal showed that edges with lengths less than $\frac{1}{n}$ of the current diameter can be shrunk without affecting the stretch of the edges cut significantly. This means that each edge only participates in $O(\log n)$ levels of recursion, and also implies that the weights given to the partition routines are within a smaller range.

If our goal are to construct Steiner trees directly, these distance estimates can be used. However, the need of generating an embeddable graph for Bartal decompositions means that we also need to return trees with small diameters in the original graph. Here, the existence of a path of much shorter edges allows us to patch the tree using parts of the minimum spanning tree. Such a tree is also implicit in previous uses of this shrinkage idea, but its explicit formulation raises the question of whether a better tree can be used.

Since our goal is to construct low stretch embeddings, a more natural choice of such a 'shortcut tree' would be one that is also low stretch. Here a natural candidate is the AKPW tree [AKPW95], since it can be computed in linear time using a bottom-up routine. By using this tree as a guide for shrinking close-by vertices, and using data structural techniques to reconstruct the original tree, we arrive our main result:

Theorem 2.8 *Let $G = (V, E, w)$ be a weighted graph with n vertices and m edges; and p any constant strictly between 0 and 1. We can construct a distribution over Bartal decompositions such that for any edge e, its expected ℓ^p-stretch in a decomposition picked from this distribution is $\mathcal{O}(\log^p n)$.*

Furthermore, we can construct a Steiner tree from a random decomposition from this distribution in $\mathcal{O}(m \log \log n)$ time in the RAM model.

3 Construction of Embeddable Trees with $\mathcal{O}(m \log n)$ ℓ_p-Stretch

We start by showing that the embedding algorithm due to Bartal [Bar96] can generate low ℓ_p-stretch embeddable trees. We first show a black-box reduction that gives embeddability by generating good decompositions under a different moment.

3.1 Embeddability Via a Different Moment

Proof of Theorem 2.6:

To prove the embeddability of $\mathcal{B}$, it is helpful to view the levels of the final Bartal decomposition as assigning weights to all the levels. For an edge e, we will use $w_i(e)$ to denote the weight of edge e in B_i.

We generate the reweighted decomposition $\mathcal{B}$ from $\mathcal{B}'$ by reweighing the edges by the levels that they are in. For an edge e that appears in B'_i, we will set its weight in B_i to:

$$w_i(e) \stackrel{\text{def}}{=} \left(\frac{\log n}{d'_i}\right)^{\frac{q-p}{p}} w(e)^{\frac{p}{q}},$$

where d'_i is the diameter bound associated with level i in $\mathcal{B}'$.

In order to show that $\mathcal{B}$ is embeddable into G, it suffices to bound the total weight assigned to e across all the layers. Here we use the fact that the d'_is are geometrically increasing, and that the levels that e appears in have $d'_i \geq l'(e)\log n$. Substituting in $l'(e) = w(e)^{-\frac{p}{q}}$ into this bound on d'_i gives:

$$\begin{aligned} d'_i &\geq w(e)^{-\frac{p}{q}} \log n \\ d'_i w(e)^{\frac{p}{q}} &\geq \log n \end{aligned}$$

Substituting this in allows us to bound the total weights assigned to the levels:

$$\begin{aligned} \sum_i w_i(e) &= \sum_{i, e\in B_i} \left(\frac{\log n}{d'_i}\right)^{\frac{q-p}{p}} w(e)^{\frac{p}{q}} \\ &= \sum_{i, e\in B_i} \left(\frac{\log n}{d'_i w(e)^{\frac{p}{q}}}\right)^{\frac{q-p}{p}} w(e) \\ &\leq \sum_{i=0}^{\infty} c^i w(e) \\ &= \mathcal{O}(w(e)), \end{aligned}$$

where c is the constant which the diameters must decrease by.

This choice of weights allows us to transfer diameter bounds from $\mathcal{B}$ to $\mathcal{B}'$. Consider the edges in B_i and B'_i and note that the length of each edge in B_i is a scaled version of its equivalent on B'_i. This allows us to set d_i by scaling d'_i by the same factor:

$$\begin{aligned} d_i &\stackrel{\text{def}}{=} \left(\frac{d'_i}{\log n}\right)^{\frac{q-p}{p}} d'_i \\ &= \log^{\frac{p-q}{p}} n \cdot {d'_i}^{\frac{q}{p}}. \end{aligned}$$

The fact that all the lengths and d'_i are scaled by the same factor means that the bound on the ratio between diameter and the length of an edge is preserved. The geometric decrease in the bounds can also be obtained from the fact that d'_i is a power of d_i times a value that only depends on n, and $d'_{i+1} \leq c d'_i$, giving $d_{i+1} \leq c^{\frac{q}{p}} d_i$.

It remains to bound the stretch of an edge e in the new decomposition using these values of d_i.

Recall that for an edge that's cut on level i, its ℓ_q-stretch w.r.t. the Bartal decomposition $\mathcal{B}'$ is

$$\mathbf{STR}^q_{\mathcal{B}'}(e) = \left(\frac{d'_i}{l'(e)}\right)^q.$$

Combining with the definition of ℓ_p and ℓ_q-stretch then gives

$$\begin{aligned}\mathbf{STR}_{\mathcal{B}}(e) &= \frac{d_i}{w(e)} \\ &= \log^{\frac{p-q}{p}} n \frac{{d'_i}^{\frac{q}{p}}}{w'(e)^{\frac{q}{p}}}, \\ \mathbf{STR}^p_{\mathcal{B}}(e) &= \log^{p-q} n \left(\frac{d'_i}{w'(e)}\right)^q = \log^{p-q} n \cdot \mathbf{STR}^q_{\mathcal{B}'}(e),\end{aligned}$$

which completes the proof. ■

3.2 Bartal's Algorithm

We now describe Bartal's algorithm, which leads to a distribution over embeddings such that the expected ℓ_p-stretch of any edge is $\mathcal{O}(\log^p n)$. This algorithm can be viewed as repeatedly partitioning a graph into pieces of smaller diameters using a probabilistic decomposition routine, whose guarantees can be stated as follows:

Lemma 3.1 (Probabilistic Decomposition) *There is an algorithm* Partition *that given a graph G with n vertices and m edges, and a diameter parameter d, returns a partition of V into $V_1 \cup V_2 \cup \ldots \cup V_k$ such that:*

1. *The diameter of the subgraph induced on each V_i is at most d with high probability, and*
2. *for any edge $e = uv$ with length $l(e)$, the probability that u and v belong to different pieces is at most $\mathcal{O}(\frac{l(e)\log n}{d})$.*

Furthermore, Partition *can be implemented using one call to finding a single source shortest path tree on the same graph with all vertices connected to a super-source by edges of length between 0 and d.*

This routine was first introduced by Bartal, and it constructed each V_i in an iterative fashion. Miller et al. [MPX13] showed that a similar procedure can be viewed globally, leading to the implementation described above. Dijkstra's algorithm (Chapter 24 of [CSRL01]) then allows one to obtain a running time of $\mathcal{O}(m+n\log n)$. It can be further sped up to $\mathcal{O}(m+n\log n)$ using Fibonacci heaps due to Fredman and Tarjan [FT87], and to $\mathcal{O}(m)$ in the RAM model by Thorup [Tho00]. In this setting where approximate answers suffice, a running time of $\mathcal{O}(m + n\log\log\Delta)$ was also obtained by Koutis et al. [KMP11]. Here we will use the $\mathcal{O}(m\log n)$ bound as it is model independent.

The decomposition is then generated by taking an upper bound on diameter, and partitioning the graphs repeatedly to ensure that its diameter decreases by factors of c at every level. Its pseudocode is given in Figure 4

It is straightforward to verify that this algorithm generates a Bartal decomposition.

$\mathcal{B} = \text{DecomposeSimple}(G, d)$, where G is a multigraph, and d is a bound on diameter.

1. Initialize $\mathcal{B}$ by setting d_0 to $2d$, and B_0 to a shortest path tree from an arbitrary vertex in V_G.
2. If G has no edges, return $\mathcal{B}$.
3. Remove all edges e with $l(e) \geq \frac{d}{c \log n}$ from G.
4. Partition G into pieces of diameter d/c, let $G_1, \ldots, G_k$ be the output of $\text{Partition}(G, d/c)$.
5. For $j = 1$ to k:
   (a) Recursively call $\text{DecomposeSimple}(G_j, d/c)$, let its output be $\mathcal{B}_j$.
   (b) Incorporate the levels of $\mathcal{B}_j$ into $\mathcal{B}$ at one lower level. Merge $B_{j,l}$ into B_{l+1} for all l implicitly.
6. Return $\mathcal{B}$.

Figure 4: Bartal's Decomposition Algorithm

Lemma 3.2 *If all edges in G have weights between $[1, \Delta]$, the output of the decomposition routine, $\mathcal{B} = \text{DecomposeSimple}(G, n\Delta)$, is, with high probability, a Bartal decomposition of G.*

Proof Since the B_is are formed by taking a union of shortest path trees on partitions of vertices, they form a forest. B_0 is also spanning because it is a spanning tree of the graph. The fact that the algorithm only refines partitions means that once two vertices are separated, they remain separated for any further partitions. This ensures that the connectivity of B_i is more than that given by any $B_{i'}$ with $i' > i$.

For the diameter bounds, the diameter bound of d ensures that the maximum distance from the starting vertex in the shortest path is bounded by d. Therefore $d_i = 2d$ suffices as a diameter bound. Since the ds are set to decrease by factors of c in all recursive calls, the values of d_i set by them are consistent, and also decreasing by factors of c. The initialization of $d = n\Delta$ ensures that no edge's length is more than $\frac{d}{\log n}$. This invariant is also kept by discarding all edges longer than $\frac{d}{c \log n}$ before each call to Partition.

■

We also need to bound the probability of cutting an edge, which can happen in two ways: either by removing it, or by cutting it in Partition.

Lemma 3.3 *For any $e \in E$ and $i \leq t$ the probability that e is cut at level i of $\mathcal{B} = \text{DecomposeSimple}(G, d)$*

is

$$\mathcal{O}\left(\frac{l(e)\log n}{d_i}\right).$$

Proof When $l(e) \geq \frac{d}{c\log n}$, a suitable choice of constants allows us to bound the probability of e being cut by 1. Otherwise, e will not be removed unless it is already cut. In case that it is in the graph passed onto PARTITION, the probability then follows from Lemma 3.1 ■

This allows us to show that each edge has low expected ℓ_p-stretch w.r.t. the resulting decomposition. In fact, we can show that a weaker condition on the cutting probability is sufficient.

Lemma 3.4 *If there is a distribution over Bartal decompositions such that the probability of an edge being cut on a level with diameter i is*

$$\mathcal{O}\left(\left(\frac{l(e)\log n}{d_i}\right)^q\right),$$

then for any $p < q$, the expected ℓ_p-stretch of an edge in a decomposition from this distribution is $\mathcal{O}(\log^p n)$.

Proof If an edge is cut at a level with $d_i \leq \log nl(e)$, its stretch is at most $\log n$, giving an ℓ_p-stretch of at most $\log^p n$. It remains to only consider the levels with $d_i \geq \log nl(e)$. Substituting the bounds of an edge cut on level i and the probability of it being cut into the definition ℓ_p-stretch gives:

$$\begin{aligned}
\mathop{\mathbf{E}}_{\mathcal{B}}\left(\mathbf{STR}^p_{\mathcal{B}}(e)\right) &\leq \sum_{i,d_i\geq\log nl(e)} \left(\frac{d_i}{l(e)}\right)^p \mathcal{O}\left(\left(\frac{l(e)\log n}{d_i}\right)^q\right) \\
&= \mathcal{O}\left(\log^p n \sum_{i,d_i\geq\log nl(e)} \left(\frac{l(e)\log n}{d_i}\right)^{q-p}\right) \\
&\leq \mathcal{O}\left(\log^p n \sum_{i=0} \left(\frac{1}{c}\right)^{q-p}\right) \\
&\leq \mathcal{O}\left(\log^p n\right).
\end{aligned}$$

Note the second last inequality follows from only considering levels with $d_i \geq l(e)\log n$ and the d_is being geometrically increasing, while the last one follows from $q - p$ and c being constants. ■

Combining these then gives the guarantees of this Bartal decomposition.

Proof of Lemma 2.7: Consider the Bartal decomposition returned by DECOMPOSESIMPLE$(G, n\Delta)$. Since the partition routine succeeds with high probability, all calls to them succeed with high probability as well. Therefore the algorithm generates a Bartal decomposition with high probability. Discarding the bad cases with high diameters only increases the bound on the probability of an edge being cut from Lemma 3.3 by a constant. Therefore invoking Lemma 3.4 with $r = 1$ gives the bound on expected ℓ_p stretch.

For running time, note that the initial diameter parameter is $d = n\Delta$, and the recursion terminates once $d < 1$. At each level, PARTITION performs a shortest path computation costing

$\mathcal{O}(m \log n)$, plus other bookkeeping operations that take linear time. The total cost over all the levels is then $\mathcal{O}(m \log(\Delta n) \log n)$ ■

4 $O(m \log \log n)$ Time Tree Construction Algorithm

We now give a faster algorithm for constructing Bartal decompositions. The algorithm proceeds in two stages. We first quickly build a lower quality decomposition using the same scheme as the AKPW low stretch spanning tree [AKPW95]. Then we proceed in the same way as Bartal's algorithm and refine the decompositions in a top-down manner. However, with the first stage decomposition, we are able to construct a Bartal decomposition much faster.

Both the AKPW decomposition and the way that our Bartal decomposition routine uses it relies on repeated clusterings of vertices. Of course, in an implementation, such clusterings will be represented using various linked-list structures. However, from an analysis perspective, it is helpful to view them as quotient graphs. For a graph G and a subset of edges A, we let the quotient graph G/A be the graph formed by the connected components of A. Each of these components corresponding to subsets of vertices becomes a single vertex in G/A, and the edges have their vertices relabeled accordingly. For our algorithms, it is essential for us to keep multi-edges as separate copies. As a result, all the graphs that we deal with in this section are potentially multi-graphs, and we will omit this distinction for simplicity.

The main advantages offered by the AKPW decomposition are

- it is a bottom-up routine that can be performed in linear time, and
- each edge only participates in $\mathcal{O}(\log \log n)$ steps of the refinement process in expectation, and
- all partition routines are done on graphs with diameter $\mathbf{poly}(\log n)$.

The interaction between the bottom-up AKPW decomposition scheme and the top-down Bartal decomposition leads to some distortions. The rest of this section can be viewed as analyzing this distortion, and the algorithmic gains from having it. We will show that for an appropriately constructed AKPW decomposition, the probability of an edge being cut can be related to a quantity in the ℓ_q norm for some $p < q < 1$. The difference between these two norms then allows us to absorb distortions of size up to $\mathbf{poly}\log n$, and therefore not affect the quality of the resulting tree. As a result, we will work mostly with a different exponent q in this section, and only bring things back to an exponent in p at the very end.

Both the AKPW and the top-down routines will issue multiple calls to Partition. In both cases the granularity of the edge weights will be $\mathbf{poly}(\log n)$. As stated in Section 3, Partition can be implemented in linear time in the RAM model, using the rather involved algorithm presented in [Tho00]. In practice, it is also possible to use the low granularity of edge weights and use Dial's algorithm [Dia69], worsening the total running time of our algorithm to $\mathcal{O}(m \log \log n + \log \Delta \ \mathbf{poly}(\log n))$ when all edge lengths are in the range $[1, \Delta]$. Alternatively, we can use the weight-sensitive shortest path algorithm from [KMP11], which works in the pointer machine model, but would be slower by a factor of $\mathcal{O}(\log \log \log n)$.

4.1 The AKPW Decomposition Routine

We first describe the AKPW algorithm for generating decomposition. The decomposition produced is similar to Bartal decompositions, although we will not impose the strict conditions on diameters in our definition.

Definition 4.1 *Let $G = (V, E, l)$ be a connected multigraph. We say that a sequence of forests $\mathcal{A}$, where*

$$\mathcal{A} = (A_0, A_1, \ldots, A_t),$$

is an AKPW decomposition *of G with parameter δ if:*

1. *A_t is a spanning tree of G.*
2. *For any $i < t$, $A_i \subseteq A_{i+1}$.*
3. *The diameter of each connected component in A_i is at most δ^{i+1}.*

Pseudocode for generating this decomposition is given in Figure 5.

$\mathcal{A} = \text{AKPW}(G, \delta)$, where G is a connected multigraph.

1. Bucket the edges by length into $E_0, E_1, \ldots$, where E_i contains all edges of length in $[\delta^i, \delta^{i+1})$
2. Initialize $A_0 := \emptyset$, $t := 0$.
3. While A_t is not a spanning tree of G:
   (a) Let E' be the set of all edges from $E_0, \ldots, E_t$ that connect different components of A_t.
   (b) Set $G_t := (V, E', \vec{1})/A_t$, where $\vec{1}$ is a constant function that assigns all edges length 1.
   (c) Decompose G by calling $\textsc{Partition}(G_t, \delta/3)$, let $T_1, T_2, \ldots, T_k$ be the edge sets of the corresponding low diameter spanning trees.
   (d) Set $A_{k+1} := A_k \cup T_1 \cup \ldots \cup T_k$
   (e) Set $t := t + 1$.
4. Return $\mathcal{A} := (A_0, \ldots, A_t)$.

Figure 5: The routine for generating AKPW decompositions

Our algorithm will use AKPW decompositions with $\delta = \log^{c'} n$ for some constant $c' > 1$ chosen based on p and q. As a result, we will treat δ as a (large) polylog factor throughout our presentation. We first bound the diameters of each piece, and the probability of an edge being cut in A_i.

Lemma 4.2 $\text{AKPW}(G, \delta)$ *generates with high probability an AKPW decomposition $\mathcal{A}$ such that for an edge $e = uv$ with $l(e) \in [\delta^i, \delta^{i+1})$ and some $j \geq i$, the probability that u and v are not connected in A_j is at most*

$$\left(\frac{c_P \log n}{\delta}\right)^{j-i},$$

where c_P is a constant associated with the partition routine. Furthermore, if $\delta \geq 2c_p \log n$, it runs in expected $\mathcal{O}(m \log\log^{1/2} n)$ time in the RAM model,

Proof The termination condition on Line 3 implies that A_t is a spanning tree, and the fact that we generate A_{i+1} by adding edges to A_i gives $A_i \subseteq A_{i+1}$. The bound on diameter can be proven inductively on i.

The base case of $i = 0$ follows from the vertices being singletons, and as a result having diameter 0. For the inductive case, suppose the result is true for i. Then with high probability each connected component in A_{i+1} corresponds to a tree with diameter $\delta/3$ connecting connected components in A_i. The definition of E_i gives that each of these edges have length at most δ^{i+1}, and the inductive hypothesis gives that the diameter of each connected component in A_i is also at most δ^{i+1}. This allows us to bound the diameter of A_{i+1} by $(\delta/3) \cdot \delta^{i+1} + (\delta/3 + 1)\delta^{i+1} \leq \delta^{i+2}$. Hence the inductive hypothesis holds for $i + 1$ as well.

The guarantees of the probabilistic decomposition routine from Lemma 3.1 gives that any an level, an edge has its two endpoints separated with probability $\frac{c_P \log n}{\delta}$. The assumption of the length of e means that it is in E_i. So by the time A_j is formed, it has gone through $j-i$ rounds of partition, and is present iff its endpoints are separated in each of these steps. Multiplying the probabilities then gives the bound.

If $\delta \geq 2c_P \log n$, then the probability of an edge in E_i appearing in subsequent levels decrease geometrically. This means that the total expected sizes of G_t processed is $O(m)$. Combining this with the linear running time of Partition gives the expected running time once we have the buckets $E_0, E_1, \ldots$. Under the RAM model of computation, these buckets can be formed in $\mathcal{O}(m \log\log^{1/2} n)$ time using the sorting algorithm by Han and Thorup [Han04]. Incorporating this cost gives the overall runtime. ■

Combining the bound on diameter and probability of an edge being cut leads to the bound on the expected ℓ_1-stretch of an edge shown by Alon et al. [AKPW95]. For an edge on the i^{th} level, the ratio between its length and the diameter of the j^{th} level can be bounded by δ^{j-i+1}. As j increases, the expected stretch of e then increases by factors of

$$\delta \cdot \mathcal{O}\left(\frac{\log n}{\delta}\right) = \mathcal{O}\left(\log n\right),$$

which leads to the more than logarithmic bound on the expected ℓ_1-stretch. With ℓ_p-stretch however, the p^{th} power of the diameter-length ratio only increases by factors of δ^p. This means that, as long as the probabilities of an edge being cut increases by factors of less than δ^p, a better bound can be obtained.

Lemma 4.3 *If $\mathcal{A}$ is generated by a call to* AKPW(G, δ) *with $\delta \geq (c_P \log n)^{\frac{1}{1-q}}$, then the probability of an edge $e \in E_i$ being cut in level j is at most $\delta^{-q(j-i)}$.*

Proof Manipulating the condition gives $c_P \log n \leq \delta^{1-q}$, and therefore

$$\left(\frac{c_P \log n}{\delta}\right)^{j-i} \leq \left(\frac{\delta^{1-q}}{\delta}\right)^{j-i} = \delta^{-q(j-i)}.$$

■

Since δ is also $\mathbf{poly}(\log n)$, we can use this bound to show that expected ℓ_p-stretch of an edge in an AKPW-decomposition can be bounded by $\mathbf{poly}(\log n)$. The exponent here can be optimized by setting q to be slightly larger than p and applying a slight variation of Lemma 3.4.

4.2 Two Pass Decomposition Algorithm

This extra factor of δ can also be absorbed into the analysis of Bartal decompositions. When $l(e)$ is significantly less than d, the difference between $\frac{l(e)\log n}{d}$ and $\left(\frac{l(e)\log n}{d}\right)^q$ is more than δ. This means that for an edge that originated much lower in the bucket, we can afford to increase its probability of being cut by a factor of δ.

From the perspective of the low-diameter decomposition routine, this step corresponds to increasing the length of an edge. This increase in length can then be used to bound the diameter of a cluster in the Bartal decomposition, and allows us to combine the bottom-up AKPW decomposition with the top-down decomposition scheme by Bartal. It also ensures that all edges that we consider have lengths close to the diameter that we partition into; and it

We start by formalizing the point at which lower levels of the AKPW decomposition can be handled at a coarser granularity. When the top-down algorithm is called with diameter d, we will use the scope to refer to this cutoff point.

Definition 4.4 *For an exponent q and a parameter $\delta \geq \log n$, we let the scope of a diameter d be*

$$scope(d) := \max_i \left\{ \delta^{i+\frac{1}{1-q}+1} \leq d \right\}.$$

Note that for small d, $scope(d)$ may be negative. As we will refer to $A_{scope(d)}$, we assume that $A_i = \emptyset$ for $i < 0$. Our full algorithm can then be viewed as only processing the edges within the scope using Bartal's top-down algorithm. Its pseudocode is given in Figure 6.

We first show that the increase in edge lengths to $\delta^{scope(d)+1}$ allows us to bound the diameter of the connected components of B_i.

Lemma 4.5 *The diameter of each connected component in B_0 is bounded by $\frac{3d}{c_D}$ with high probability.*

Proof By the guarantee of the partition routine, the diameter of each G'_i is at most $\frac{d}{c_D}$. However, since we are measuring diameter of the components in G, we also need to account for the diameter of the components that were shrunken into vertices when forming G'. These components corresponds to connected pieces in $A_{scope(d)}$, therefore the diameters of the corresponding trees are bounded by $\delta^{scope(d)+1}$ with high probability. Our increase of edge weights in G', on the other hand, ensures that the length of any edge is more than the diameter of its endpoints. Hence the total increase in diameter from these pieces is at most twice the length of a path in G', and the diameter of these components in G can be bounded by $\frac{3d}{c_D}$. ■

The increase in edge lengths leads to increases in the probabilities of edges being cut. We next show that because the AKPW decomposition is computed using a higher norm, this increase can be absorbed, giving a probability that is still closely related to the p^{th} power of the ratio between the current diameter and the length of the edge.

$\mathcal{B}$ = DECOMPOSERECURSIVE$(G, d, p, q, \mathcal{A})$, where G is a graph, d is the current diameter, p, q are exponents, and $\mathcal{A}$ is a fixed AKPW decomposition.

1. If G has no edges, return () and quit.
2. Adjust d and the level so that $A_{scope(d)}$ does not span G.
3. Let G' be $G/A_{scope(d)}$ with all edge lengths increased to at least $\delta^{scope(d)+1}$.
4. Remove all edges e with $l(e) \geq \frac{d}{c_D \log n}$ from G'.
5. Partition G' into pieces of diameter d/c_D, let $G'_1, \ldots, G'_k$ be the output of PARTITION$(G', d/c_D)$.
6. Set B_0 to the shortest path trees in $G'_1, \ldots, G'_k$ unioned with $A_{scope(d)}$ (implicitly).
7. For $j = 1$ to k:
   (a) Recursively call DECOMPOSESIMPLE$(G_j, 3d/c_D)$, let its output be $\mathcal{B}_j$.
   (b) Incorporate the levels of $\mathcal{B}_j$ into $\mathcal{B}$ at one lower level. Add $B_{j,l}$ to B_{l+1} for all l implicitly.
8. Return $(B_0, \ldots, B_t)$.

Figure 6: Pseudocode of two pass algorithm for finding a Bartal decomposition

Lemma 4.6 *Assume $\mathcal{A} = \text{AKPW}(G, \delta)$. For any edge e with length $l(e)$ and any level i, the probability that e is cut at level i of $\mathcal{B}$ is*

$$\mathcal{O}\left(\left(\frac{l(e)\log n}{d_i}\right)^q\right).$$

Proof There are two cases to consider based whether the length of the edge is more than $\delta^{scope(d)+1}$. If it is and it appears in G', then its length is retained. The guarantees of PARTITION then gives that it is cut with probability

$$\mathcal{O}\left(\frac{l(e)\log n}{d_i}\right) \leq \mathcal{O}\left(\left(\frac{l(e)\log n}{d_i}\right)^q\right),$$

where the inequality follows from $l(e)\log n \leq d_i$.

Otherwise, since we contracted the connected components in $A_{scope(d)}$, the edge is only cut at level i if it is both cut in $A_{scope(d_i)}$, and its corresponding edge in G' is cut by the partition routine. Lemma 4.3 gives that if the edge is from E_j, its probability of being cut in $A_{scope(d_i)}$ can be bounded

by $\delta^{-q(scope(d_i)-j)}$. Combining with the fact that $\delta^j \leq l(e)$ allows us to bound this probability by

$$\left(\frac{l(e)}{\delta^{scope(d_i)}}\right)^q.$$

Also, since the weight of the edge is set to $\delta^{scope(d_i)+1}$ in G', its probability of being cut by PARTITION is

$$\mathcal{O}\left(\frac{\delta^{scope(d_i)+1}\log n}{d_i}\right).$$

As the partition routine is independent of the AKPW decomposition routine, the overall probability can be bounded by

$$\mathcal{O}\left(\frac{\delta^{scope(d_i)+1}\log n}{d_i}\cdot\left(\frac{l(e)}{\delta^{scope(d_i)}}\right)^q\right)=\mathcal{O}\left(\left(\frac{l(e)\log n}{d_i}\right)^q\cdot\delta\log^{1-q}n\cdot\left(\frac{\delta^{scope(d_i)}}{d_i}\right)^{1-q}\right).$$

Recall from Definition 4.4 that $scope(d_i)$ is chosen to satisfy $\delta^{scope(d_i)+\frac{1}{1-q}+1}\leq d_i$. This along with the assumption that $\delta \geq \log n$ gives

$$\delta\log^{1-q}n\cdot\left(\frac{\delta^{scope(d_i)}}{d_i}\right)^{1-q}\leq\delta^{2-q}\left(\delta^{-\frac{2-q}{1-q}}\right)^{1-q}\leq 1.$$

Therefore, in this case the probability of e being cut can also be bounded by $\mathcal{O}\left(\left(\frac{l(e)\log n}{d_i}\right)^q\right)$. ■

Therefore, we can still obtain the properties of a good Bartal decomposition by only considering edges in the scope during the top-down partition process. On the other hand, this shrinking drastically improves the performance of our algorithm.

Lemma 4.7 *Assume $\mathcal{A}$ = AKPW(G,δ). For any edge e, the expected number of recursive calls to* DECOMPOSERECURSIVE *in which e is included in the graph given to* PARTITOIN *can be bounded by $\mathcal{O}(\log\log n)$.*

Proof Note that for any level i it holds that

$$\delta^{scope(d_i)}\geq d_i\delta^{-\frac{1}{1-q}-2}.$$

Since the diameters of the levels decrease geometrically, there are at most $\mathcal{O}(\log\log n)$ levels i such that $l(e)\in[d_i\delta^{-\frac{1}{1-q}-2},\frac{d_i}{\log n})$.

The expected number of occurrences of e in lower levels can be bounded using Lemma 4.3 in a way similar to the proof of the above Lemma. Summing over all the levels i where e is in a lower level gives:

$$\sum_{i:l(e)<d_i\delta^{-\frac{1}{1-q}-2}}\left(\frac{l(e)}{\delta^{scope(d_i)}}\right)^q$$

Substituting in the bound on $\delta^{scope(d_i)}$ from above and rearranging then gives:

$$\leq \sum_{i:l(e)\leq d_i\delta^{-\frac{1}{1-q}-2}} \left(\frac{l(e)}{d_i}\delta^{\frac{1}{1-q}+2}\right)^q.$$

As d_i increase geometrically, this is a geometric sum with the first term at most 1. Therefore the expected number of times that e appears on some level i while being out scope is $\mathcal{O}(1)$. ■

Recall that each call to PARTITION runs in time linear in the number of edges. This then implies a total cost of $\mathcal{O}(m \log\log n)$ for all the partition steps. We can now proceed to extract a tree from this decomposition, and analyze the overall runtime cost.

4.3 Returning a Tree

We now give the overall algorithm and analyze its performance. Introducing the notion of scope in the recursive algorithm limits each edge to appear in at most $\mathcal{O}(\log\log n)$ levels. Each of these calls partitions G' in time linear in its size, which should give a total of $\mathcal{O}(m \log\log n)$. However, the goal of the algorithm as stated is to produce a Bartal decomposition, which has a spanning tree at each level. Explicitly generating this gives a total size of $\Omega(nt)$, where t is the number of recursive calls. As a result, we will circumvent this by storing only an implicit representation of the Bartal decomposition to find the final tree.

This smaller implicit representation stems from the observation that large parts of the B_is are trees from the AKPW decomposition, A_i. As a result, such succinct representations are possible if we have pointers to the connected components of A_i. We first analyze the quality and size of this implicit decomposition, and the running time for producing it.

$\mathcal{B}$ = DECOMPOSE(G, p), where G is a graph, p is an exponent

1. Set $q = \frac{1+p}{2}$, $\delta = (c \log n)^{\frac{1}{q-p}}$.
2. Compute an AKPW decomposition of G, $\mathcal{A} = \text{AKPW}(G, \delta)$.
3. D be the diameter of A_t.
4. Set $\mathcal{B}$:= DECOMPOSERECURSIVE$(G, D, p, q, \mathcal{A})$.
5. Set B_0 to A_t.
6. Return $\mathcal{B}$.

Figure 7: Overall decomposition algorithm

Lemma 4.8 *There is a routine* DECOMPOSE *that for any graph G and and constant $p < 1$, produces in expected $\mathcal{O}(m \log\log n)$ time an implicit representation of a Bartal decomposition $\mathcal{B}$ with size $\mathcal{O}(m \log\log n)$ and consisting of edges and weighted connected components of an AKPW decomposition of G such that with high probability*

- *$\mathcal{B}$ is embeddable into G, and*

- *for any edge e,* $\mathbf{E}_{\mathcal{B}}(\boldsymbol{STR}^p_{\mathcal{B}}(e)) \leq \mathcal{O}(\log^p n)$

Proof Consider the routine given in Figure 7. In the RAM model, bucketing the edges and computing the AKPW decomposition can be done in $\mathcal{O}(m \log\log n)$ time. The resulting tree can be viewed as a laminar decomposition of the graph. This is crucial for making the adjustment of d in $\mathcal{O}(1)$ time to ensure that $A_{scope(d)}$ is disconnected. This in turn allows the cost of each recursive call to be bounded by the number of edges in G'. As each edge participates in at most $\mathcal{O}(m \log\log n)$ G's, the total cost can be bounded by $\mathcal{O}(m \log\log n)$.

Since the number of AKPW components implicitly referred to at each level of the recursive call is bounded by the total number of vertices, and in turn the number of edges, the total number of such references is bounded by the size of the G's as well. This gives the bound on the size of the implicit representation.

We then apply EmbeddableDecompose to use this routine to generate an embeddable decomposition. The transformation of the edge weights can clearly be performed in linear time. Once a decomposition is computed on this transformed graph, Theorem 2.6 gives that it suffices to rescale each level uniformly. This process affects the implicit decomposition by changing the weights of the AKPW pieces, which is can be done in $\mathcal{O}(1)$ time since our output allows for such parameters. ■

It remains to show that a tree can be generated efficiently from this implicit representation. To do this, it is enough to find contractions (as defined in Definition 2.3) of the trees of the AKPW decomposition to the corresponding sets of connecting endpoints. Here we use the fact that the AKPW decomposition is in fact a single tree.

Fact 4.9 *Let $\mathcal{A} = A_0, \ldots, A_t$ be an AKPW decomposition of G. Let S be a subset of vertices of G. For any i in $\{0, \ldots, t\}$, if S is contained in a single connected component of A_i, then the contraction of A_i to S is equal to the contraction of A_t to S.*

This allows us to use data structures to find the contractions of the AKPW trees to the respective vertex sets more efficiently.

Lemma 4.10 *Given a tree A_t on the vertex set V (with $|V| = n$) and subsets $S_1, \ldots, S_k$ of V of total size $\mathcal{O}(n)$, we can generate the contractions of A_t to each of the sets S_i in time $\mathcal{O}(n)$ in the RAM model and $\mathcal{O}(n\alpha(n))$ in the pointer machine model.*

Proof Root A_t arbitrarily. Note that the only explicit vertices required in the contraction of A_t to a set $S \subseteq V$ are

$$\Gamma(S) \stackrel{\text{def}}{=} S \cup \{LCA(u, v) : u, v \in S\}$$

where $LCA(u, v)$ denotes the lowest common ancestor of u and v in A_t. Moreover, it is easily verified that if we sort the vertices $v_1, \ldots, v_{|S|}$ of S according to the depth first search pre-ordering, then

$$\Gamma(S) = S \cup \{LCA(v_i, v_{i+1}) : 1 \leq i < |S|\}.$$

We can therefore find $\Gamma(S_i)$ for each i simultaneously in the following steps:

1. Sort the elements of each S_i according to the pre-ordering, using a single depth-first search traversal of A_t.

2. Prepare a list of lowest common ancestor queries for each pair of vertices adjacent in the sorted order in each set S_i.

3. Answer all the queries simultaneously using an off-line lowest common ancestor finding algorithm.

Since the total number of queries in the last step is $\mathcal{O}(n)$, its runtime is $\mathcal{O}(n\alpha(n))$ in the pointer machine model using disjoint union [Tar79], and $\mathcal{O}(n)$ in the RAM model [GT83].

Once we find the sets $\Gamma(S_i)$ for each i, we can reconstruct the contractions of A_t as follows:

1. Find the full traversal of the vertices in $\Gamma(S_i)$ for each i, using a single depth first search traversal of A_t.

2. Use this information to reconstruct the trees [Vui80].

■

Proof of Theorem 2.8: Consider the distribution over Bartal decompositions given by Lemma 4.8. We will apply the construction given in Lemma 2.5, albeit in a highly efficient manner.

For the parts of the decomposition that are explicitly given, the routine runs in linear time. The more intricate part is to extract the smaller contractions from the AKPW components that are referenced to implicitly. Since all levels of the AKPW decomposition are subtrees of A_t, these are equivalent to finding contractions of A_t for several sets of vertices, as stated in Fact 4.9. The algorithm given in Lemma 4.10 performs this operation in linear time. Concatenating these trees with the one generated from the explicit part of the decomposition gives the final result. ■

5 Sufficiency of Embeddability

In the construction of our trees, we made a crucial relaxation of only requiring embeddability, rather than restricting to subgraphs. In this section, we show that linear operators on the resulting graph can be related to linear operators on the original graph. Our analysis is applicable to ℓ_∞ flows as well.

The spectral approximation of two graphs can be defined in terms of their Laplacians. As we will interpret these objects combinatorially, we omit their definition and refer the reader to Doyle and Snell [DS84]. For matrices, we can define a partial ordering $\preceq$ where $A \preceq B$ if $B - A$ is positive semi-definite. That is, for any vector $\mathbf{x}$ we have

$$\mathbf{x}^T A\mathbf{x} \leq \mathbf{x}^T B\mathbf{b}.$$

If we let the graph formed by adding the tree to G be H, then our goal is to bound L_G and L_H with each other. Instead of doing this directly, it is easier to relate their pseudo-inverses. This will be done by interpreting $\mathbf{x}^T L^\dagger \mathbf{x}$ in terms of the energy of electrical flows. The energy of an electrical flow is defined as the sum of squares of the flows on the edges multiplied by their resistance, which in our case equals the length of the edge. Given a flow $f \in \Re^E$, we will denote its electrical energy

using

$$\mathcal{E}_G(f) \stackrel{\text{def}}{=} \sum_e l_e f(e)^2.$$

The residue of a flow f is the net in/out flow at each vertex. This give a vector on all vertices, and finding the minimum energy of flows that meet a given residue is equivalent to computing $\mathbf{x}^T L^\dagger \mathbf{x}$. The following fact plays a central role in the monograph by Doyle and Snell [DS84]:

Fact 5.1 *Let G be a connected graph. For any vector $\boldsymbol{x}$ orthogonal to the all ones vector, $\boldsymbol{x}^T L_G^\dagger \boldsymbol{x}$ equals the minimum electrical energy of a flow with residue $\boldsymbol{x}$.*

Lemma 5.2 *Let $G = (V_G, E_G, w_G)$ and $H = (V_H, E_H, w_H)$ be graphs such that G is a subgraph of H in the weighted sense and $H \setminus G$ is embeddable in G. Furthermore, let the graph Laplacians of G and H be L_G and L_H respectively. Also, let Π be the $|V_G| \times |V_H|$ matrix with one 1 in each row at the position that vertex corresponds to in H and 0 everywhere else, and Π_1 the orthogonal projection operator onto the part of $\Re^{V_G}$ that's orthogonal to the all-ones vector. Then we have:*

$$\frac{1}{2} L_G^\dagger \preceq \Pi_1 \Pi L_H^\dagger \Pi^T \Pi_1^T \preceq L_G^\dagger.$$

Proof Since $\Pi_1^T = \Pi_1$ projects out any part space spanned by the all ones vector, and is this precisely the null space of L_G, it suffices to show the result for all vectors $\mathbf{x}_G$ orthogonal to the all-1s vector. These vectors are in turn valid demand vectors for electrical flows. Therefore, the statement is equivalent to relating the minimum energies of electrical flows routing $\mathbf{x}_G$ on G and $\Pi^T \mathbf{x}_G$ on H.

We first show that flows on H take less energy than the ones in G. Let $\mathbf{x}_G$ be any vector orthogonal to the all ones vector, and f_G^* be the flow of minimum energy in G that meets demand $\mathbf{x}_G$. Setting the same flow on the edges of $E(G)$ in H and 0 on all other edges yields a flow f_H. The residue of this flow is the same residue in V_G, and 0 everywhere else, and therefore equal to $\Pi^T \mathbf{x}_G$. Since G is a subgraph of H in the weighted sense, the lengths of these edges can only be less. Therefore the energy of f_H is at most the energy of f_G and we have

$$\mathbf{x}_G^T \Pi L_H^\dagger \Pi^T \mathbf{x}_G \leq \mathcal{E}_H(f_H) \leq \mathcal{E}_G(f_G^*) = \mathbf{x}_G^T L_G^\dagger \mathbf{x}_G.$$

For the reverse direction, we use the embedding of $H \setminus G$ into G to transfer the flow from H into G. Let $\mathbf{x}_G$ be any vector orthogonal to the all ones vector, and f_H^* the flow of minimum energy in H that has residue $\Pi^T \mathbf{x}_G$. This flow can be transformed into one in G that has residue $\mathbf{x}_G$ using the embedding. Let vertex/edge mapping of this embedding be π_V and π_E respectively.

If an edge $e \in E_H$ is also in E_G, we keep its flow value in G. Otherwise, we route its flow along the path that the edge is mapped to. Formally, if the edge is from u to v, $f_H(e)$ units of flow is routed from $\pi_V(u)$ to $\pi_V(v)$ along $path(e)$. We first check that the resulting flow, f_G has residue $\mathbf{x}_G$. The net amount of flow into a vertex $u \in V_G$ is

$$\sum_{uv \in E_G} f_H^*(e) + \sum_{u'v' \in E_H \setminus E_G, \pi_V(u') = u} f_H^*(e) = \sum_{uv \in E_G} f_H^*(e) + \sum_{u' \in V_H, \pi_V(u') = u} \left(\sum_{u'v' \in E_H \setminus E_G} f_H^*(e) \right).$$

Reordering the summations and noting that $\Pi(u) = u$ gives

$$= \sum_{u' \in V_H, \pi_V(u') = u} \sum_{u'v' \in E_H} f_H(e) = \sum_{u' \in V_H, \pi_V(u') = u} \left(\Pi^T \mathbf{x}_G\right)(e) = x_G(u).$$

The last equality is because $\pi_V(u) = u$, and all vertices not in V_G having residue 0 in $\Pi^T \mathbf{x}_G$.

To bound the energy of this flow, the property of the embedding gives that if split the edges of G into the paths that form the embedding, each edge is used at most once. Therefore, if we double the weights of G, we can use one copy to support G, and one copy to support the embedding. The energy of this flow is then the same. Hence there is an electrical flow f_G in G such that $\mathcal{E}_G(f_G) \leq 2\mathcal{E}_H(f_H^*)$. Fact 5.1 then gives that it is an upper bound for $\mathbf{x}_G^T L_G^\dagger \mathbf{x}_G$, completing the proof. ■

Acknowledgments

We thank Ittai Abraham, Anupam Gupta, Nick Harvey, Jon Kelner, Yiannis Koutis, Aaron Sidford, Dan Spielman, and Zeyuan Zhu for very helpful comments and discussions.

References

[ABN08] Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. In *Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '08, pages 781–790, Washington, DC, USA, 2008. IEEE Computer Society. 1.1

[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. 1.1, 2, 4, 4.1

[AN12] Ittai Abraham and Ofer Neiman. Using petal decompositions to build a low stretch spanning tree. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 395–406, New York, NY, USA, 2012. ACM. 1.1, 1.2

[Bar96] Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 184–193, 1996. 1.1, 2, 2, 2, 3

[Bar98] Yair Bartal. On approximating arbitrary metrices by tree metrics. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 161–168, New York, NY, USA, 1998. ACM. 1.1

[BGK+11] Guy E. Blelloch, Anupam Gupta, Ioannis Koutis, Gary L. Miller, Richard Peng, and Kanat Tangwongsan. Near linear-work parallel SDD solvers, low-diameter decomposition, and low-stretch subgraphs. In *Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures*, SPAA '11, pages 13–22, New York, NY, USA, 2011. ACM. 1

[BGK+13] Guy E. Blelloch, Anupam Gupta, Ioannis Koutis, GaryL. Miller, Richard Peng, and Kanat Tangwongsan. Nearly-linear work parallel SDD solvers, low-diameter decomposition, and low-stretch subgraphs. *Theory of Computing Systems*, pages 1–34, March 2013. 1, 1.2

[CFM+14] Micheal Cohen, Brittany Terese Fasy, Gary L. Miller, Amir Nayyeri, Richard Peng, and Noel Walkington. Solving 1-laplacians of convex simplicial complexes in nearly linear time: Collapsing and expanding a topological ball. 2014. To appear in SODA 2014. 1

[CKM+11] Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 273–282, New York, NY, USA, 2011. ACM. 1

[CKP+13] Michael B. Cohen, Rasmus Kyng, Jakub Pachocki, Richard Peng, and Anup Rao. Preconditioning in expectation. STOC'14 submission #219, 2013. 1.2

[CSRL01] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001. 3.2

[Dia69] Robert B. Dial. Algorithm 360: shortest-path forest with topological ordering [h]. *Commun. ACM*, 12(11):632–633, November 1969. 4

[DS84] Peter G. Doyle and J. Laurie Snell. *Random Walks and Electric Networks*, volume 22 of *Carus Mathematical Monographs*. Mathematical Association of America, 1984. 5

[DS08] Samuel I. Daitch and Daniel A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, pages 451–460, New York, NY, USA, 2008. ACM. 1

[EEST08] Michael Elkin, Yuval Emek, Daniel A Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. *SIAM Journal on Computing*, 38(2):608–628, 2008. 1.1

[FRT04] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, November 2004. 1.1

[FT87] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34:596–615, July 1987. 3.2

[Gre96] Keith D. Gremban. *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems*. PhD thesis, Carnegie Mellon University, 1996. 1

[GT83] Harold N. Gabow and Robert Endre Tarjan. A linear-time algorithm for a special case of disjoint set union. In *Proceedings of the 15th annual ACM symposium on Theory of computing (STOC)*, pages 246–251, New York, NY, USA, 1983. ACM. 4.3

[Han04] Yijie Han. Deterministic sorting in o(nlog logn) time and linear space. *J. Algorithms*, 50(1):96–105, January 2004. 4.1

[KLOS13] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. *CoRR*, abs/1304.2338, 2013. 1

[KMP11] Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-m log n time solver for SDD linear systems. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 590–598, Washington, DC, USA, 2011. IEEE Computer Society. 1, 1.2, 2, 2, 3.2, 4

[KOSZ13] Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving SDD systems in nearly-linear time. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, STOC '13, pages 911–920, New York, NY, USA, 2013. ACM. 1

[LS13] Yin Tat Lee and Aaron Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. *CoRR*, abs/1305.1922, 2013. 1.2

[Mad10] Aleksander Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *FOCS*, pages 245–254. IEEE Computer Society, 2010. 1.2

[Mad13] Aleksander Madry. Navigating central path with electrical flows: from flows to matchings, and back. *CoRR*, abs/1307.2205, 2013. 1

[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, 3.2

[PS13] Richard Peng and Daniel A. Spielman. An efficient parallel solver for SDD linear systems. STOC'14 submission #216, 2013. 1.2

[She13] Jonah Sherman. Nearly maximum flows in nearly linear time. *CoRR*, abs/1304.2077, 2013. 1, 1.2

[Sid13] Aaron Sidford. personal communication, 2013. 1.2

[ST04] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 81–90, June 2004. 1

[Tar79] Robert Endre Tarjan. Applications of path compression on balanced trees. *J. ACM*, 26(4):690–715, 1979. 4.3

[Tho00] Mikkel Thorup. Floats, integers, and single source shortest paths. *Journal of Algorithms*, 35:189–201, 2000. 3.2, 4

[Vai91] Pravin M. Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. A talk based on this manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation, October 1991. 1

[Vui80] Jean Vuillemin. A unifying look at data structures. *Commun. ACM*, 23(4):229–239, April 1980. 2