# Parallel Probabilistic Tree Embeddings, k-Median, and Buy-at-Bulk Network Design

Guy E. Blelloch Anupam Gupta Kanat Tangwongsan
Carnegie Mellon University
{guyb, anupamg, ktangwon}@cs.cmu.edu

## ABSTRACT

This paper presents parallel algorithms for embedding an arbitrary $n$-point metric space into a distribution of dominating trees with $O(\log n)$ expected stretch. Such embedding has proved useful in the design of many approximation algorithms in the sequential setting. We give a parallel algorithm that runs in $O(n^2 \log n)$ work and $O(\log^2 n)$ depth—these bounds are independent of $\Delta = \frac{\max_{x,y} d(x,y)}{\min_{x \neq y} d(x,y)}$, the ratio of the largest to smallest distance. Moreover, when $\Delta$ is exponentially bounded ($\Delta \leq 2^{O(n)}$), our algorithm can be improved to $O(n^2)$ work and $O(\log^2 n)$ depth.

Using these results, we give an RNC $O(\log k)$-approximation algorithm for $k$-median and an RNC $O(\log n)$-approximation for buy-at-bulk network design. The $k$-median algorithm is the first RNC algorithm with non-trivial guarantees for arbitrary values of $k$, and the buy-at-bulk result is the first parallel algorithm for the problem.

**Categories and Subject Descriptors:** F.2 [Theory of Computation]: Analysis of Algorithms and Problem Complexity

**General Terms:** Algorithms, Theory

**Keywords:** Parallel algorithms, probabilistic tree embedding, k-median, buy-at-bulk network design

## 1. INTRODUCTION

The idea of embedding a finite metric into a distribution of "simpler" metrics has proved to be a useful and versatile technique in the algorithmic study of metric spaces, with far-reaching consequences to understanding finite metrics and developing approximation algorithms. An important line of work in this pursuit is concerned with embedding a finite metric space into a distribution of dominating trees [Bar98] that minimizes distance distortion: Given an $n$-point metric space $(X, d)$, the goal is to find a distribution $\mathcal{D}$ of trees to minimize $\beta$ such that

(1) *Non-contracting:* for all $T \in \mathcal{D}$, $d(x, y) \leq d_T(x, y)$ for all $x, y \in X$; and

(2) *Small Distortion:* $\mathbf{E}_{T \sim \mathcal{D}}[d_T(x, y)] \leq \beta \cdot d(x, y)$ for all $x, y \in X$,

where $d_T(\cdot, \cdot)$ denotes the distance in the tree $T$, and $\mathbf{E}_{T \sim \mathcal{D}}$ draws a tree $T$ from the distribution $\mathcal{D}$. The series work on this problem (e.g., [Bar98, Bar96, AKPW95]) culminated in the result of Fakcharoenphol, Rao, and Talwar (FRT) [FRT04], who gave an elegant optimal algorithm with $\beta = O(\log n)$. This has proved to be instrumental in many approximation algorithms (see, e.g., [FRT04] and the references therein). For example, the first polylogarithmic approximation algorithm for the $k$-median problem was given using this embedding technique. Remarkably, all these algorithms only require pairwise expected stretch, so their approximation guarantees are expected approximation bounds and the technique is only applicable to problems whose objectives are linear in the distances.

In this paper, motivated by getting a parallel algorithm for the $k$-median problem, we consider the problem of computing such tree embeddings in parallel. None of the previous low-depth parallel approximation algorithms for $k$-median worked for all ranges of $k$; all previous algorithms giving non-trivial approximations either had more than polylogarithmic depth, or could only handle $k$ smaller than $\text{polylog}(n)$ [BT10]. Using the parallel tree-embedding results in this paper, we give parallel approximation algorithms for $k$-median. Moreover, we use these embeddings to give parallel approximation algorithms for the *buy-at-bulk network design* problem.

A crucial design constraint is to ensure that the parallel work of our algorithms remains close to that of the sequential counterparts (aka. *work efficiency*) while achieving small, preferably polylogarithmic, depth (parallel time). Work efficiency is important since it allows an algorithm to be applied efficiently to both a modest number of processors (*one* being the most modest) and a larger number. Even with a larger number of processors, work-efficient algorithms limit the amount of resources used and hence presumably the cost of the computation (e.g. in terms of the energy used, or the rental cost of a machine in the "cloud"). We will be less concerned with polylogarithmic factors in the depth since such a measure is typically not robust across models.

**Our Results.** We give a parallel algorithm to embed any $n$-point metric into a distribution of hierarchically well-separated trees (HSTs)[1] with $O(n^2 \log n)$ (randomized) work and $O(\log^2 n)$ depth that offers the same distance-preserving guarantees as FRT [FRT04]. When the ratio between the largest and smallest distance $\Delta = \frac{\max_{x,y} d(x,y)}{\min_{x \neq y} d(x,y)}$ is exponentially bounded (i.e., $\Delta \leq 2^{O(n)}$), our algorithm can be improved to $O(n^2)$ work and $O(\log^2 n)$ depth. The main challenge arises in ensuring the depth of the computation is polylogarithmic even when the resulting tree is highly imbalanced—the FRT algorithm, as stated, works level by level, and a naïve

[1] A definition of HSTs is given in Section 2.


*SPAA'12,* June 25–27, 2012, Pittsburgh, Pennsylvania, USA.

implementation incurs $O(\log \Delta)$ depth, which is undesirable when $\Delta$ is more than polynomial in $n$. Our contribution lies in recognizing an alternative view of the FRT algorithm and developing an efficient algorithm to exploit it. Our analysis also implies probabilistic embeddings into trees without Steiner nodes of height $O(\log n)$ whp. (though not HSTs).

Using these embedding algorithms, in Section 4, we give an RNC $O(\log k)$-approximation for $k$-median—see Theorem 4.4. This is the first RNC algorithm that gives non-trivial approximation for arbitrary values of $k$. (There is an RNC algorithm that give a $(5+\varepsilon)$-approximation for $k \leq \text{polylog}(n)$ [BT10].) Furthermore, the algorithm is work-efficient relative to previously described sequential techniques. It remains an intriguing open problem to give an RNC algorithm for $k$-median with a constant-factor approximation.

Finally, in Section 5, we give an RNC $O(\log n)$-approximation algorithm for buy-at-bulk network design—see Theorem 5.1. This algorithm for buy-at-bulk network design is within an $O(\log n)$ factor of being work efficient if the input contains all-pairs shortest paths distances of the graph. All our algorithms are randomized and all our guarantees are expected approximation bounds.

## 2. PRELIMINARIES AND NOTATION

Throughout the paper, let $[n] = \{1, 2, \ldots, n\}$. For alphabet $\Sigma$ and a sequence $\alpha \in \Sigma^*$, we denote by $|\alpha|$ the length of $\alpha$ and by $\alpha_i$ (or alternatively $\alpha(i)$) the $i$-th element of $\alpha$. Given sequences $\alpha$ and $\beta$, we say that $\alpha \sqsubseteq \beta$ if $\alpha$ is a prefix of $\beta$. Furthermore, we denote by $\text{LCP}(\alpha, \beta)$ the *longest common prefix* of $\alpha$ and $\beta$. Let $\text{prefix}(\alpha, i)$ be the first $i$ elements of $\alpha$.

Let $G = (V, E)$ be a graph with edge lengths $\ell : E \to \mathbb{R}_+$. Let $d_G(u, v)$ or simply $d(u, v)$ denote the shortest-path distance in $G$ between $u$ and $v$. We write $V(G)$ and $E(G)$ to mean the vertex set and the edge set of $G$. We represent graphs and trees in a form of adjacency array, where the vertices and the edges are each stored contiguously, and each vertex has a pointer to a contiguous array of pointers to its incident edges. A *hierarchically well-separated tree* (HST) is a rooted tree where the edges from each node to its children have the same length, and the lengths of consecutive edges on any root-leaf path decrease by some factor $\alpha > 1$.

An event happens with high probability (**whp.**) if it happens with probability at least $1 - n^{-\Omega(1)}$. We analyze the algorithms in the PRAM model and use both the EREW (Exclusive Read Exclusive Write) and CRCW (Concurrent Read Concurrent Write), assuming for the CRCW that an arbitrary value is written. By *work*, we mean the total operation count, and by *depth*, we mean the longest chain of dependencies (i.e., parallel time).

A *trie* (also known as a prefix tree) is an ordered tree where each tree edge is marked with a symbol from (constant-sized) $\Sigma$ and a node $v$ corresponds to the sequence given by the symbols on the root-to-$v$ path, in that order. In this work, we only deal with non-empty sequences $s_1, s_2, \ldots, s_k$ of equal length. The trie corresponding to these sequences is one in which there are $k$ leaf nodes, each corresponding uniquely to one of the sequences. If these sequences have long common prefixes, its trie has many long paths (i.e., a line of non-branching nodes ). This can be compressed. Contracting all non-root degree-2 nodes by concatenating the symbols on the incident edges results in a *Patricia tree* (also known as a radix tree), in which by definition, all internal node except the root has degree at least 3. Using a (multiway-)Cartesian tree algorithm of Blelloch and Shun and a known reduction [BS11], the Patricia tree of a lexicographically ordered sequence of strings $s_1, \ldots, s_n$ can be constructed in $O(n)$ work and $O(\log^2 n)$ depth assuming the following as input: (1) the sequences $s_i$'s themselves, (2) $|s_i|$ the length of each $s_i$ for $i \in [n]$, and (3) $|\text{LCP}(s_i, s_{i+1})|$ the length of the longest common prefix between $s_i$ and $s_{i+1}$ for $i \in [n-1]$.

We also rely on the following primitives on trees. Given a commutative semigroup $(U, *)$, and a rooted tree $T$ (not necessarily balanced) where every node $v \in V(T)$ is tagged with a value $\text{val}(v) \in U$, there is an algorithm `treeAgg` that computes the aggregate value for each subtree of $T$ (i.e., for each $v \in V(T)$, compute the result of applying $*$ to all $\text{val}(\cdot)$ inside that subtree) in $O(n)$ work and $O(\log n)$ depth, assuming the binary operator $*$ is a constant-time operation [MRK88, JáJ92]. In the same work-depth bounds, the lowest common ancestor (LCA) of a pair of vertices $u$ and $v$, denoted by $\text{LCA}(u, v)$, can be determined (via the tree primitive just mentioned or otherwise) [SV88, BV93].

Finally, we recall a useful fact about random permutations:

**Lemma 2.1 ([Sei92])** *Let $\pi : [n] \to [n]$ be a random permutation on $[n]$, and let $y_i = \min\{\pi(j) : j = 1, \ldots, i\}$. Then, $y_i$'s form a non-increasing sequence. Moreover, the number of times $y_i$'s change to a smaller value is expected $O(\log n)$. In fact, it is $O(\log n)$ with high probability.*

## 3. PARALLEL FRT EMBEDDING

An input instance is a finite metric space $(X, d)$, where $|X| = n$ and the symmetric distance function $d(\cdot, \cdot)$ is specified by an $n$-by-$n$ matrix, normalized so that for all $x \neq y$, $1 \leq d(x, y) \leq \Delta$, where $\Delta$ is a power of two (i.e., $\Delta = 2^\delta$). As is standard, we assume that $d(x, x) = 0$.

In the sequential case, FRT [FRT04] developed an elegant algorithm that preserves the distances up to $O(\log n)$ in expectation. Their algorithm can be described as a top-down recursive *low-diameter decomposition* (LDD) of the metric. In broad strokes, the algorithm is given a metric space $(X, d)$ with diameter $\Delta$ and it applies an LDD procedure to partition the points into clusters of diameter roughly $\Delta/2$, then each cluster into smaller clusters diameter of $\Delta/4$, etc. This construction produces a laminar family of clusters that we connect up based on set-inclusion, yielding a so-called FRT tree. The algorithm gives an optimal distance-preserving guarantee and as described it can be implemented in $O(n^2 \log n)$ sequential time if $\Delta \leq 2^{O(n)}$.

While the low-diameter decomposition step is readily parallelizable, there is potentially a long chain of dependencies in the tree construction: for each $i$, determining the clusters with diameter $2^i$ requires knowing the clusters of diameter $2^{i+1}$. This $O(\log \Delta)$ chain is undesirable for large $\Delta$. Our algorithms get rid of this dependence. The main theorem of this section is the following:

**Theorem 3.1 (Parallel FRT Embedding)** *There is a randomized algorithm that on input a finite metric space $(X, d)$ with $|X| = n$, produces a tree $T$ such that for all $x, y \in X$, $d(x, y) \leq d_T(x, y)$ (with probability 1) and $\mathbf{E}[d_T(x, y)] \leq O(\log n)\, d(x, y)$. The algorithm runs in $O(n^2 \log n)$ work and $O(\log^2 n)$ depth **whp.** Furthermore, if $\Delta \leq 2^{O(n)}$, the algorithm can be improved to run in $O(n^2)$ work and $O(\log^2 n)$ depth **whp.***

**Remarks.** When $\Delta \leq 2^{O(n)}$, the improvement comes from replacing comparison-based sort with parallel radix sort; we describe this in more detail in the algorithm's description.

### 3.1 FRT Tree as Sequences

To achieve this parallelization, we take a different, though equivalent, view of the FRT algorithm. The main conceptual difference is as follows. Instead of adopting a cluster-centric view which maintains

**Algorithm 3.1** Implicit simultaneous low-diameter decompositions

1. Pick a permutation $\pi : X \to [n]$ uniformly at random.
2. Pick $\beta \in [1, 2]$ with the distribution $f_\beta(x) = 1/(x \ln 2)$.
3. For all $v \in X$, compute the partition sequence $\chi^{(v)}_{\pi,\beta}$.

a set of clusters that are refined over time, we explore a point-centric view which tracks the movement of each point across clusters but without explicitly representing the clusters. This view can also be seen as representing an FRT tree by the root-to-leaf paths of all external nodes (corresponding to points in the metric space). We formalize this idea in the following definition:

**Definition 3.2 ($(\pi, \beta)$-Partition Sequence)** *For $v \in X$, the* partition sequence *of $v$ with respect to a permutation $\pi : X \to [n]$ and a parameter $\beta > 0$, denoted by $\chi^{(v)}_{\pi,\beta}$, is a length-$(\delta + 1)$ sequence such that $\chi^{(v)}_{\pi,\beta}(0) = 1$ and*

$$\chi^{(v)}_{\pi,\beta}(i) = \min\{\pi(w) \mid w \in X,\ d(v, w) \le \beta \cdot 2^{\delta - i - 1}\}$$

*for $i = 1, \ldots, \delta = \log_2 \Delta$.*

For each combination of $\pi$ and $\beta$, each $\chi^{(v)}_{\pi,\beta}$ is the sequence of the lowest-numbered vertices (where the numbering is given by the random permutation $\pi$) that the node $v$ can "see" as it reduces its range-of-vision geometrically from $\Delta$ down to 0. Thus, these numbers keep increasing from $1 = \min_{w \in X} \pi(w)$ to $\pi(v)$. Hence, the first step in generating an FRT tree is to pick a random permutation $\pi$ on the nodes and a value $\beta$, and compute the partition sequence $\chi^{(v)}_{\beta,\pi}$ for each node $v \in X$, as shown in Algorithm 3.1. These partition sequences encode all the information we need to construct an FRT tree $T$, which can be done as follows:

*Vertices:* For $i = 0, \ldots, \delta$, let $L_i = \{\mathsf{prefix}(\chi^{(v)}_{\pi,\beta}, i+1) \mid v \in X\}$ be the $i$-th level in the tree. The vertices of $T$ are exactly $V(T) = \cup_i L_i$, where each $v \in X$ corresponds to the node identified by the sequence $\chi^{(v)}_{\pi,\beta}$.

*Edges:* Edges only go between $L_i$ and $L_{i+1}$ for $i \ge 1$. In particular, a node $x \in L_i$ has an edge with length $2^{\delta - i}$ to $y \in L_{i+1}$ if $x \sqsubseteq y$.

This construction yields a tree because edges are between adjacent levels and defined by the subsequence relation. Note that the full $\chi^{(v)}_{\pi,\beta}$ are the leaves of $T$.

We will show in the following two lemmas distance-preserving properties of the tree $T$. The first lemma shows that $d_T$ is an upper bound for $d$; the second shows that $d_T$ preserves the distance $d$ up to a $O(\log n)$ factor in expectation.

**Lemma 3.3** *For all $u, v \in X$, for all $\beta, \pi$,*

$$d(u, v) \ \le \ d_T(\chi^{(u)}_{\pi,\beta}, \chi^{(v)}_{\pi,\beta}).$$

PROOF. The proof is straightforward and is given for completeness. Let $u, v \in X$ such that $u \ne v$ be given. These nodes are "separated" at a vertex $y$ that is the longest common prefix (LCP) of $\chi^{(u)}$ and $\chi^{(v)}$. Let $i^* = |\text{LCP}(\chi^{(u)}, \chi^{(v)})|$. This means there is a vertex $w$ at distance at most $\beta \cdot 2^{\delta - i^* - 1}$ from both $u$ and $v$, so $d(u, v) \le 2^{\delta - i^* + 1}$. On the other hand, both $\chi^{(u)}$ and $\chi^{(v)}$ are in the subtree rooted at $y$; therefore, $d_T(\chi^{(u)}, \chi^{(v)}) \ge 2 \cdot 2^{\delta - i^*} \ge 2^{\delta - i^* + 1}$, which concludes the proof. □

**Lemma 3.4** *For all $u, v \in X$,*

$$\mathbf{E}\left[d_T(\chi^{(u)}_{\pi,\beta}, \chi^{(v)}_{\pi,\beta})\right] \ \le \ O(\log n) \cdot d(u, v).$$

This process is equivalent to that of [FRT04]. Here, we adapt their proof to our setting. Before beginning the analysis, we state a useful fact: Because $\beta$ is picked from $[1, 2]$ with the probability density function (pdf.) $\frac{1}{x \ln 2}$, we have

$$\mathbf{Pr}\left[\exists i \ge 1, \beta \cdot 2^{i-1} \in [x, x + dx)\right] \le \tfrac{dx}{x \ln 2}. \qquad (3.1)$$

PROOF. Let distinct $u, v \in X$ be given. In the tree constructed, $\chi^{(u)}$ and $\chi^{(v)}$ will be separated at the least common ancestor (LCA) of the two nodes. At the split point, the two sequences differ for the first time. For the analysis, we will need two definitions: First, we say that $\chi^{(u)}$ and $\chi^{(v)}$ are *split* by $w \in X$ if $\pi(w)$ is the smaller of the two values at the first position where $\chi^{(u)}$ and $\chi^{(v)}$ differ. Second, we say that $w \in X$ is a *lead* of a vertex $u$ if $\pi(w) = \min\{\pi(z) \mid d(u, z) \le d(u, w)\}$ (i.e., $w$ has the smallest $\pi$ number among points in the ball of radius $d(u, w)$ centered at $u$).

The crux of the argument is in noticing that $u$ and $v$ are necessarily split by some $w$, and when this happens, the following must be true:

1. There exists a level $i$ such that $d(w, u) \le \beta \cdot 2^{\delta - i - 1} < d(w, v)$ —we assume without loss of generality that $d(w, u) \le d(w, v)$.
2. $w$ is a lead of $u$ or $v$. (Note that $w$ must be a lead since it is the smaller of the two values at the position where the sequences first differ.)

For each node $w$, define $\texttt{contrib}_w$ to be the distance in the tree between $\chi^{(u)}$ and $\chi^{(v)}$ assuming $w$ splits $u$ and $v$. The following claim bounds $\texttt{contrib}_w$ conditioned on $w$ being a lead of $u$ or $v$:

**Claim 3.5** *For any $w \ne u, v$,*

$$\mathbf{E}\left[\texttt{contrib}_w \mid w \text{ is a lead of } u \text{ or } v\right] \ \le \ 8d(u, v).$$

Using this claim, which will be proved below, we can express $\mathbf{E}\left[d_T(\chi^{(u)}, \chi^{(v)})\right]$ as follows: Because we know $d_T(\chi^{(u)}, \chi^{(v)}) \le \sum_w \texttt{contrib}_w$, we have

$$\begin{aligned}\mathbf{E}\left[d_T(\chi^{(u)}, \chi^{(v)})\right] \ &\le \ \sum_w \mathbf{E}\left[\texttt{contrib}_w \mid w \text{ is a lead of } u, v\right] \\ &\qquad \times \mathbf{Pr}\left[w \text{ is a lead of } u, v\right] \\ &\le \ 8d(u, v)\, \mathbf{E}\left[\sum_w \mathbf{1}_{\{w \text{ is a lead of } u, v\}}\right].\end{aligned}$$

To complete the proof, we know from Lemma 2.1 that the number of vertices that could be a lead of $u$ or $v$ is $O(\log n)$ in expectation. Therefore, we conclude that $\mathbf{E}\left[d_T(\chi^{(u)}, \chi^{(v)})\right] \le O(\log n) d(u, v)$. □

PROOF OF CLAIM 3.5. Assume WLOG that $d(w, u) < d(w, v)$. By (3.1), for a particular $x$ between $d(w, u)$ and $d(w, v)$, we know

$$\mathbf{Pr}\left[\exists i \ge 1, \beta \cdot 2^{\delta - i - 1} \in [x, x + dx)\right] \le \frac{dx}{x \ln 2}.$$

Furthermore, if $u$ is separated from $v$ at level $i$, then

$$\texttt{contrib}_w \le 2 \cdot \sum_{i' \ge i} 2^{\delta - i'} = 2^{\delta - i + 2} \le 8\beta 2^{\delta - i - 1}.$$

This means that the claimed expectation is at most

$$\int_{d(w,u)}^{d(w,v)} \frac{1}{x \ln 2}\, 8x\, dx \leq 8(d(w,v) - d(w,u)) \leq 8d(u,v),$$

which concludes the proof. □

## 3.2 A Simple Parallel FRT Algorithm

We now present a naïve parallelization of the above construction. This naïve version still has the $\log \Delta$ dependence. Notice that a parallel algorithm with such parameters can be inferred directly from [FRT04]; however, the presentation here is instructive: it relies on computing partition sequences and building the tree using them, which will be useful for the improved parallel algorithm in the section that follows.

**Lemma 3.6** *Given $\pi$ and $\beta$, each $\chi^{(v)}_{\pi,\beta}$ can be computed in (worst-case) $O((n + \log \Delta) \log n)$ work and $O(\log n)$ depth.*

PROOF. Let $v \in X$, together with $\pi$ and $\beta$, be given. We can sort the vertices by the distance from $v$ so that $v = v_n$ and $d(v, v_1) \geq d(v, v_2) \geq \cdots \geq d(v, v_n) = 0$, where $v_1, \ldots, v_n$ are distinct vertices. This requires $O(n \log n)$ work and $O(\log n)$ depth. Then, we compute $\ell_i = \min\{\pi(v_j) \mid j \geq i\}$ for $i = 1, \ldots, n$. This quantity indicates that by going to distance at least $d(v, v_i)$, the point $v$ could reach a number as low as $\ell_i$. This step requires $O(n)$ work and $O(\log n)$ depth (using a prefix computation). Finally, for each $k = 1, \ldots, \delta$, use a binary search to determine the smallest index $i$ (i.e., largest distance) such that $d(v, v_i) \leq \beta \cdot 2^{\delta - k - 1}$—and $\chi^{(v)}(k)$ is simply $\ell_i$. There are $O(\log \Delta)$ such $k$ values, each independently running in $O(\log n)$ depth and work, so this last step requires $O(\log \Delta \log n)$ work and $O(\log n)$ depth, which completes the proof. □

Using this algorithm, we can compute all partition sequences independently in parallel, leading to a total of $O(n(n + \log \Delta) \log n)$ work and $O(\log n)$ depth for computing $\chi^{(v)}$ for all $v \in X$. The next step is to derive an embedding tree from these partition sequences. From the description in the previous section, to compute the set of level-$i$ vertices, we examine all length-$i$ prefixes $\mathsf{prefix}(\chi^{(v)}_{\pi,\beta}, i)$ for $v \in X$ and remove duplicates. The edges are easy to derive from the description. Each level $i$ can be done in expected $O(i^2)$ work and $O(\log n)$ depth, so in total we need $O(\log^3 \Delta)$ work and $O(\log n \log \Delta)$ depth in expectation to build the tree from these sequences, proving the following theorem:

**Theorem 3.7** *There is an algorithm* `simpleParFRT` *that computes an FRT tree in expected $O(n^2 \log n + n \log \Delta \log n + \log^3 \Delta)$ work and $O(\log n \log \Delta)$ depth.*

Notice that the naïve algorithm has $\Delta$ dependence in both work and depth. While this is fine for small $\Delta$, the depth term can be undesirable for large $\Delta$. For example, when $\Delta$ is $2^{O(n)}$, the algorithm has $O(n \log n)$ depth. In the section that follows, we present an algorithm that removes this $\Delta$ dependence.

## 3.3 An Improved Algorithm

The simple algorithm in the preceding section had a $\log \Delta$ dependence in both work and depth. We now show the power of the partition sequence view and derive an algorithm whose work and depth bounds are independent of $\Delta$. Moreover, the algorithm performs the same amount of work as the sequential algorithm.

At first glance, the $\log \Delta$ dependence in the generation of partition sequences in our previous algorithm seems necessary, and the reason is simple: the length of each partition sequence is $O(\log \Delta)$. To remove this dependence, we work with a different representation of partition sequences, one which has length at most $n$ in the worst case but with high probability, has length $O(\log n)$. This representation is based on the observation that any partition sequence is non-decreasing and its entries are numbers between 1 and $n$. Consequently, the sequence cannot change values more than $n$ times and we only have to remember where it changes values. Furthermore, by Lemma 2.1, we know that this sequence changes values at most $O(\log n)$ times **whp.** This inspires the following definition:

**Definition 3.8 (Compressed Partition Sequence)** *For $v \in X$, the* compressed partition sequence *of $v$, denoted by $\sigma^{(v)}_{\pi,\beta}$, is the unique sequence $\langle (s_i, p_i) \rangle_{i=1}^{k}$ such that $1 = s_1 < \cdots < s_k < s_{k+1} = \delta + 1$, $p_1 < p_2 < \cdots < p_k$, and for all $i \leq k$ and $j \in \{s_i, s_i + 1, \ldots, s_{i+1} - 1\}$, $\chi^{(v)}_{\pi,\beta}(j) = p_i$, where $\chi^{(v)}_{\pi,\beta}$ is the partition sequence of $v$.*

In words, if we view $\pi$ as assigning priority values to $X$, then the compressed partition sequence of $v$ tracks the distance scales at which the lowest-valued vertex within reach from $v$ changes. As an example, at distances $\beta \cdot 2^{\delta - s_1 + 1}, \beta \cdot 2^{\delta - (s_1+1) + 1}, \ldots, \beta \cdot 2^{\delta - (s_2 - 1) + 1}$, the lowest-valued vertex within reach of $v$ is $p_1$—and $\beta \cdot 2^{\delta - s_2 + 1}$ is the first distance scale at which $p_1$ cannot be reached and $p_2 > p_1$ becomes the new lowest-valued node. The following lemma shows how to efficiently compute the compressed partition sequence of a given vertex.

**Lemma 3.9** *Given $\pi$ and $\beta$, each compressed partition sequence $\sigma^{(v)}_{\pi,\beta}$ can be computed in (worst-case) $O(n \log n)$ work and $O(\log n)$ depth. Furthermore, if $\Delta \leq 2^{O(n)}$, this can be improved to $O(n)$ work and $O(\log n)$ depth* ***whp.***

PROOF. The idea is similar to that of the partition sequence, except for how we derive the sequence at the end. Let $v \in X$, together with $\pi$ and $\beta$, be given. Sort the vertices by the distance from $v$ so that $v = v_n$ and $d(v, v_1) \geq d(v, v_2) \geq \cdots \geq d(v, v_n)$, where $v_1, \ldots, v_n$ are distinct vertices. This has $O(n \log n)$ work and $O(\log n)$ depth—or, if $\Delta \leq 2^{O(n)}$, this can be done in $O(n)$ work and $O(\log n)$ depth since the distance scales then can be identified by numbers between 0 and $O(n)$, which can be efficiently sorted using parallel radix sort [RR89]. Again, compute $\ell_i = \min\{\pi(v_j) \mid j \geq i\}$ for $i = 1, \ldots, n$. Furthermore, let $b_i = \max\{j \geq 1 \mid \beta \cdot 2^{\delta - j - 1} \geq d(v, v_i)\}$ for all $i = 1, \ldots, n$. This index $b_i$ represents the smallest distance scale that $v$ can still see $v_i$. Then, we compute $\rho_i = \min\{\ell_j \mid b_j = b_i\}$. Because the $b_i$'s are non-decreasing, computing $\rho_i$'s amounts to identifying where $b_i$'s change values and performing a prefix computation. Thus, the sequences $\ell_i$'s, $b_i$'s, and $\rho_i$'s can be computed in $O(n)$ work and $O(\log n)$ depth.

To derive the compressed partition sequence, we look for all indices $i$ such that $\rho_{i-1} \neq \rho_i$ and $b_{i-1} \neq b_i$—these are precisely the distance scales at which the current lowest-numbered vertex becomes unreachable from $v$. These indicies can be found in $O(n)$ work and $O(1)$ depth, and using standard techniques involving prefix sums, we put them next to each other in the desired format. □

We will now construct an FRT tree from these compressed partition sequences. Notice that to keep the work term independent of $\log \Delta$, we cannot, for example, explicitly write out all the nodes. The FRT tree has to be in a specific "compressed" format for the construction to be efficient. For this reason, we will store the resulting FRT tree in a compressed format. A *compacted FRT tree* is

obtained by contracting all degree-2 internal nodes of an FRT tree, so that every internal node except for the root has degree at least 3 (a single parent and at least 2 children). By adding the weights of merged edges, the compacting preserves the distance between every pair of leaves. Equivalently, an FRT tree as described earlier is in fact a trie with the partition sequences as its input—and a compacted FRT tree is a *Patricia (or radix) tree on these partition sequences*.

Our task is therefore to construct a Patricia tree given compressed partition sequences. As discussed in Section 2, the Patricia tree of a *lexicographically ordered* sequence of strings $s_1, \ldots, s_n$ can be constructed in $O(n)$ work and $O(\log^2 n)$ depth if we have the following as input: (1) $|s_i|$ the length of each $s_i$ for $i \in [n]$, and (2) $\text{LCP}(s_i, s_{i+1})$ the length of the longest common prefix between $s_i$ and $s_{i+1}$ for $i \in [n-1]$. These sequences can be lexicographically ordered in no more than $O(n \log^2 n)$ work[2] and $O(\log^2 n)$ depth **whp.** and the LCP between all adjacent pairs can be computed in $O(n^2)$ work and $O(\log n)$ depth. Combining this with the Patricia tree algorithm [BS11] gives the promised bounds, concluding the proof of Theorem 3.1.

## 3.4 FRT Tree Without Steiner Vertices

Some applications call for a tree embedding solution that consists of only the original input vertices. To this end, we describe how to convert a compacted FRT tree from the previous section into a tree that contains no Steiner vertices. As a byproduct, the resulting non-Steiner tree has $O(\log n)$ depth with high probability.

**Theorem 3.10** *There is an algorithm* `FRTNoSteiner` *running in $O(n^2 \log n)$ work and $O(\log^2 n)$ depth **whp.** that on input an $n$-point metric space $(X, d)$, produces a tree $T$ such that (1) $V(T) = X$; (2) $T$ has $O(\log n)$ depth **whp.**; and (3) for all $x, y \in X$, $d(x, y) \leq d_T(x, y)$ and $\mathbf{E}[d_T(x, y)] \leq O(\log n)\, d(x, y)$. Furthermore, the bounds can be improved to $O(n^2)$ work and $O(\log^2 n)$ depth **whp.** if $\Delta \leq 2^{O(n)}$.*

We begin by recalling that each leaf of an FRT tree corresponds to a node in the original metric space, so there is a bijection $f : L_\delta \to X$. Now consider an FRT tree $T$ on which we will perform the following transformation: (1) obtain $T'$ by multiply all the edge lengths of $T$ by 2, (2) for each node $x \in V(T')$, label it with $\mathsf{label}(x) = \min\{\pi(f(y)) \mid y \in \mathsf{leaves}(T'_x)\}$, where $\mathsf{leaves}(T'_x)$ is the set of leaf nodes in the subtree of $T'$ rooted at $x$, and (3) construct $T''$ from $T'$ by setting an edge to length 0 if the endpoints are given the same label—but retaining the length otherwise. The following lemma bounds the depth of the resulting tree $T''$ (the proof appears in the appendix).

**Lemma 3.11** *The depth of $T''$ is $O(\log n)$ with high probability.*

Several things are clear from this transformation: First, for all $u, v \in X$, $d(u, v) \leq d_{T'}(\chi^{(u)}, \chi^{(v)})$ and

$$\mathbf{E}\left[d_{T'}(\chi^{(u)}, \chi^{(v)})\right] \leq O(\log n)\, d(u, v)$$

(of course, with worse constants than $d_T$). Second, $T''$ is no longer an HST, but $d_{T''}$ is a lowerbound on $d_{T'}$ (i.e., for all $u, v \in X$, $d_{T''}(\chi^{(u)}, \chi^{(v)}) \leq d_{T'}(\chi^{(u)}, \chi^{(v)})$). Therefore, to prove distance-preserving guarantees similar to Theorem 3.1, we only have to show that $d_{T''}$ dominates $d$ (the proof appears in Appendix A).

**Lemma 3.12** *For all $u, v \in X$, $d(u, v) \leq d_{T''}(\chi^{(u)}, \chi^{(v)})$.*

[2]The length of the compressed partition sequence of $v$ is upperbounded by the number of times the sequence $y_i = \min\{\pi(v_j) \mid n-1 \geq j \geq i\}$ changes value. By Lemma 2.1, this is at most $O(\log n)$ **whp**.

On a compacted FRT tree, this transformation is easy to perform. First, we identify all leaves with their corresponding original nodes. Computing the label for all nodes can be done in $O(n)$ work and $O(\log n)$ depth using `treeAgg` (Section 2), which is a variant of tree contraction. Finally, we just have to contract zero-length edges, which again, can be done in $O(n)$ work and $O(\log n)$ depth using standard tree-contraction techniques [JáJ92]. Notice that we only have to compute the minimum on the nodes of a compacted tree, because the label (i.e., the minimum value) never changes unless the tree branches.

# 4. THE K-MEDIAN PROBLEM

The $k$-median problem is a standard clustering problem, which has received considerable attention from various research communities. The input to this problem is a set of vertices $V \subseteq X$, where $(X, d)$ is a (finite) metric space, and the goal is to find a set of at most $k$ centers $F_S \subseteq V$ that minimizes the objective

$$\Phi(F_S) \quad = \quad \sum_{j \in V} d(j, F_S).$$

Since we will be working with multiple metric spaces, we will write $\Phi_D(F_S) = \sum_{j \in V} D(j, F_S)$ to emphasize which distance function is being used. In the sequential setting, several approximation algorithms are known, including $O(1)$-approximations (see [AGK+04] and the references therein) and approximation via tree embeddings [Bar98, FRT04]. In the parallel setting, these algorithms seem hard to parallelize directly: to our knowledge, the only RNC algorithm for $k$-median gives a $(5+\varepsilon)$-approximation but only achieves polylogarithmic depth when $k$ is at most $\text{polylog}(n)$ [BT10].

Our goal is to obtain an $O(\log k)$-approximation that *has polylogarithmic depth for all $k$* and has essentially the same work bound as the sequential counterpart. The basic idea is to apply bottom-up dynamic programming to solve $k$-median on a tree, like in Bartal's paper [Bar98]. Later, we describe a sampling procedure to improve the approximation guarantee from $O(\log n)$ to $O(\log k)$. While dynamic programming was relatively straightforward to apply in the sequential setting, more care is needed in the parallel case: the height of a compacted FRT tree can be large, and since the dynamic program essentially considers tree vertices level by level, the total depth could be much larger than $\text{polylog}(n)$.

Rather than working with compacted FRT trees, we will be using FRT trees that contain no Steiner node, constructed by the algorithm `FRTNoSteiner` in Theorem 3.10. This type of trees is shown to have the same distance-preserving properties as an FRT tree but has $O(\log n)$ depth with high probability. Alternatively, we give an algorithm that reduces the depth of a compacted FRT tree to $O(\log n)$; this construction, which assumes the HST property, is presented in Appendix B and may be of independent interest.

## 4.1 Solving k-Median on Trees

Our second ingredient is a parallel algorithm for solving $k$-median when the distance metric is the shortest-path distance in a (shallow) tree. For this, we will parallelize a dynamic programming (DP) algorithm of Tamir [Tam96], which we now sketch. Tamir presented a $O(kn^2)$ algorithm for a slight generalization of $k$-median on trees, where in his setting, each node $i \in V$ is associated with a cost $c_i$ if it were to be chosen; every node $i$ also comes equipped with a nondecreasing function $f_i$; and the goal becomes to find a set $A \subseteq V$ of size at most $k$ to minimize

$$\sum_{i \in A} c_i + \sum_{j \in V} \min_{i \in A} f_j(d(v_j, v_i)).$$

This generalization (which also generalizes the facility-location problem) provides a convenient way of taking care of Steiner nodes in an FRT tree. For our purpose, these $f_i$'s will simply be the identity function[3] $x \mapsto x$, and $c_i$'s are set so that it is 0 if $i$ is a real node from the original instance and $\infty$ if $i$ is a Steiner node (to prevent it from being chosen).

Tamir's algorithm is a DP which solves the problem exactly, but it requires the input tree to be binary. While Tamir also gave an algorithm that converts any tree into a binary tree, his approach can significantly increase the depth of the tree. For our setting, *we need a different algorithm that ensures not only that the tree is binary but also that the depth does not grow substantially.* The simplest solution is to replace each node that has outdegree $d$ with a perfect binary tree with $d$ leaves; this can increase the depth of the tree from $O(\log n)$ to $O(\log^2 n)$ in the worst case. But this increase can be avoided. We give a parallel algorithm based on the Shannon-Fano code construction [Sha48], as detailed in Section 4.2 below, which outputs a binary tree whose depth is an additive $\log n$ larger than the original depth. Then, we set edge lengths as follows: The new edges will have length 0 except for the edges incident to the original $v_i$'s: the parent edge incident to $v_i$ will inherit the edge length from $v_i v$. Also, the added nodes have cost $\infty$. As a result of this transformation, the depth of the new tree is at most $O(\log n)$ and the number of nodes will at most double.

The main body of Tamir's algorithm begins by producing for each node $v$ a sequence of vertices that orders all vertices by their distances from $v$ with ties appropriately broken. His algorithm for generating these sequences are readily parallelizable because the sequence for a vertex $v$ involves merging the sequences of its children in a manner similar to the merge step in merge sort. The work for this step, as originally analyzed, is $O(n^2)$. Each merge can be done in $O(\log n)$ depth and there are at most $O(\log n)$ levels; this step has depth $O(\log^2 n)$.

## 4.2 Making Trees Binary

Given a tree $T$ with $n$ nodes and depth $h$ but of arbitrary fanout, we give a construction that yields a tree with depth $O(h + \log n)$. In contrast, the simpler alternative that replaces each node that has fanout $d$ with a perfect binary tree with $d$ leaves produces a tree with depth $O(h \log n)$ in the worst case. Instead, our algorithm makes use of a well-known idea, which goes back to a construction of Shannon and Fano [Sha48]: For each original tree node $v$ with children $v_1, \ldots, v_k$, we assign to $v_i$ the "probability"

$$p_i = \frac{|V(T_{v_i})|}{|V(T_v)|},$$

where $T_u$ denotes the subtree of $T$ rooted at $u$. Shannon and Fano's result implies that there is a binary tree whose external nodes are exactly these $v_1, \ldots, v_k$, and the path from $v$ to $v_i$ in this binary tree has length at most $1 + \log(1/p_i)$. Applying this construction on all nodes with degree more than 2 gives the following lemma:

**Lemma 4.1** *Given a tree $T$ with $n$ nodes and depth $h$ but of arbitrary fanout, there is an algorithm* `treeBinarize` *producing a binary tree with depth at most $h + \log n$.*

PROOF. Let $w$ be a leaf node in $T$ and consider the path from the root node to $w$. Suppose on this path, the subtrees have sizes $n = n_1 > n_2 > \cdots > n_{d'} = 1$, where $d' \le h$. Applying the aforementioned construction, this path expands to a path of length at most

$$1 + \sum_{i=1}^{d'-1} \left(1 + \log(\tfrac{1}{p_i})\right) = 1 + \sum_{i=1}^{d'-1} \left(1 + \log(\tfrac{n_i}{n_{i+1}})\right)$$
$$\le d' + \log n$$
$$\le h + \log n,$$

which proves the lemma. □

This construction is also directly parallelizable as all that is needed is sorting the probabilities in decreasing order (so that $p_1 \ge p_2 \ge \cdots p_k$), computing the cumulative probabilities (i.e., $P_i = \sum_{i' \le i} p_i$), finding the point where the cumulative probability splits in (roughly) half, and recursively applying the algorithm on the two sides. Since each call involves a sort and a prefix computation, and the total number of children is $O(n)$, the transformation on the whole tree will take $O(n \log n)$ work and $O(\log^2 n)$ depth.

## 4.3 Paralellizing the DP Step

Armed with this, the actual DP is straightforward to parallelize. Tamir's algorithm maintains two tables $F$ and $G$, both indexed by a tuple $(i, q, r)$ where $i \in [n]$ represents a node, $q \le k$ counts the number of centers inside the subtree rooted at $i$, and $r$ indicates roughly the distance from $i$ to the closest selected center outside of the subtree rooted at $i$. As such, for each $i$ and $q$, there can be at most $n$ different values for $r$. Now Tamir's DP is amendable to parallelization because the rules of the DP compute an entry using *only* the values of its immediate children. Further, each rule is essentially taking the minimum over a combination of parameters and can be parallelized using standard algorithms for finding the minimum and prefix sums. Therefore, we can compute the table entries for each level of the tree and move on to a higher level. It is easy to show that each level can be accomplished in $O(\log n)$ depth, and as analyzed in Tamir's paper, the total work is bounded by $O(kn^2)$.

## 4.4 Parallel Successive Sampling

Our algorithm thus far gives an $O(\log n)$-approximation on input consisting of $n$ points. To improve it to $O(\log k)$, we devise a parallel version of Mettu and Plaxton's successive sampling (shown in Algorithm 4.1) [MP04]. We then describe how to apply it to our problem. Since the parallel version produces an identical output to the sequential one, guarantees about the output follow directly from the results of Mettu and Plaxton. Specifically, they showed that there are suitable settings of $\alpha$ and $\beta$ such that by using $K = \max\{k, \log n\}$, the algorithm $\texttt{SuccSampling}_{\alpha,\beta}(V, K)$ runs for $O(\log(n/K))$ rounds and produces $Q$ of size at most $O(K \cdot \log(n/K))$ with the following properties:

**Theorem 4.2 (Mettu and Plaxton [MP04])** *There exists an absolute constant $C_{SS}$ such that if $Q =$* `SuccSampling` $(V, K)$*, then with high probability, $Q$ satisfies $\Phi(Q) \le C_{SS} \cdot \Phi(OPT_k)$, where $OPT_k$ is an optimal $k$-median solution on the instance $V$.*

In other words, the theorem says that $Q$ is a bicriteria approximation which uses $O(K \log(n/K))$ centers and obtains a $C_{SS}$-approximation to $k$-median. To obtain a parallel implementation of successive sampling (Algorithm 4.1), we will make steps 1–3 parallel. We have the following runtime bounds:

**Lemma 4.3** *For $|V| = n$ and $K \le n$,* `SuccSampling`$(V, K)$ *has $O(nK)$ work and $O(\log^2 n)$ depth* ***whp.***

[3] In the weighted case, we will use $f_i(x) = w_i \cdot x$ to reflect the weight on node $i$.

**Algorithm 4.1** $\texttt{SuccSampling}_{\alpha,\beta}(V, K)$—successive sampling

Let $U_0 = V, i = 0$
**while** $(|U_i| > \alpha K)$
1. Sample from $U_i$ u.a.r. (with replacement) $\lfloor \alpha K \rfloor$ times—call the chosen points $S_i$.
2. Compute the smallest $r_i$ such that $|B_{U_i}(S_i, r_i)| \geq \beta \cdot |U_i|$ and let $C_i = B_{U_i}(S_i, r_i)$, where $B_U(S, r) = \{w \in U \mid d(w, S) \leq r\}$.
3. Let $U_{i+1} = U_i \setminus C_i$ and $i = i + 1$.

Output $Q = S_0 \cup S_1 \cup \cdots \cup S_{i-1} \cup U_i$

PROOF. First, by the choice of $C_i$, we remove at least a $\beta$ fraction of $U_i$ and since $\alpha$ and $\beta$ are constants, we know that the number of iterations of the **while** loop is $O(\log(n/K))$. Now step 1 of the algorithm can be done in $O(nK)$ work and $O(1)$ depth (assuming concurrent writes). To perform Step 2, first, we compute for each $p \in U_i$, the distance to the nearest point in $S_i$. This takes $O(|U_i|K)$ work and $O(\log K)$ depth. Then, using a linear-work selection algorithm, we can find the set $C_i$ and $r_i$ in $O(|U_i|)$ work and $O(\log |U_i|)$ depth. Since each time $|U_i|$ shrinks by a factor $\beta$, the total work is $O(nK)$ and the total depth is $O(\log^2 n)$. □

Piecing together the components developed so far, we obtain an expected $O(\log k)$-approximation. The following theorem summarizes our main result for the $k$-median problem:

**Theorem 4.4** *For $k \geq \log n$, there is a randomized algorithm for the $k$-median problem that produces an expected $O(\log k)$-approximate solution running in $O(nk + k(k \log(\frac{n}{k}))^2) \leq O(kn^2)$ work and $O(\log^2 n)$ depth* ***whp.*** *For $k < \log n$, the problem admits an expected $O(1)$-approximation with $O(n \log n + k^2 \log^5 n)$ work and $O(\log^2 n)$ depth* ***whp.***

Here is a proof sketch, see Appendix A.1 for more details: we first apply Algorithm 4.1 to get the set $Q$ (in $O(nK)$ work and $O(\log^2 n)$ depth, Lemma 4.3). Then, we "snap" the clients to their closest centers in $Q$ (paying at most $C_{SS}\,\Phi(\text{OPT}_k)$ for this), and depending on the range of $k$, either use an existing parallel $k$-median algorithm for $k < \log n$ [BT10] or use the FRT-based algorithm on these "moved" clients to get the $O(\log q)$-approximation (in $O(kq^2)$ work and $O(\log^2 q)$ depth, where $q = O(K \log(n/K))$, because we are running the algorithm only on $O(K \log(n/K))$ points). Note that we now need a version of the $k$-median algorithm on trees (Section 4.1) where clients also have weights, but this is easy to do (by changing the $f_i$'s to reflect the weights).

# 5. BUY-AT-BULK NETWORK DESIGN

In this section, we explore another application of our parallel probabilistic tree embedding algorithm. Let $G = (V, E)$ be an undirected graph with $n$ nodes; edge lengths $\ell : E \to \mathbb{R}_+$; a set of $k$ demand pairs $\{\mathsf{dem}_{s_i,t_i}\}_{i=1}^k$; and a set of cables, where cable of type $i$ has capacity $u_i$ and costs $c_i$ per unit length. The goal of the problem is to find the cheapest set of cables that satisfy the capacity requirements and connect each pair of demands by a path. Awerbuch and Azar [AA97] gave a $O(\log n)$-approximation algorithm in the sequential setting. Their algorithm essentially finds an embedding of the shortest-path metric on $G$ into a distribution of trees *with no Steiner nodes*, a property which they exploit when assigning each tree edge to a path in the input graph. For an edge with net demand $\mathsf{dem}$, the algorithm chooses the cable type that minimizes $c_i\lceil \mathsf{dem}/u_i \rceil$. This is shown to be an expected $O(\log n)$-approximation. From a closer inspection, their algorithm can be parallelized by developing parallel algorithms for the following:

1. Given a graph $G$ with edge lengths $\ell : E(G) \to \mathbb{R}_+$, compute a dominating tree $T$ with no Steiner node such that $T$ $O(\log n)$-probabilistically approximate $d$ in expectation.
2. For each $(u, v) \in E(T)$, derive the shortest path between $u$ and $v$ in $G$.
3. For each $e \in E(T)$, compute the net demand that uses this edge, i.e., $f_e = \sum_{i:e \in P_T(s_i,t_i)} \mathsf{dem}_{s_i,t_i}$, where $P_T(u, v)$ denotes the unique path between $u$ and $v$ in $T$.

We consider these in turn. First, the shortest-path metric $d$ can be computed using a standard all-pair shortest paths algorithm in $O(n^3 \log n)$ work and $O(\log^2 n)$ depth. With this, we apply the algorithm in Section 3.4 to produce a dominating tree $T$ in which $d_T$ $O(\log n)$-probabilistically approximates $d$. Furthermore, for each tree edge $e = (u, v)$, the shortest path between $u$ and $v$ can be obtained from the calculation performed to derive $d$ at no extra cost.

Next we describe how to calculate the net demand on every tree edge. We give a simple parallel algorithm using the `treeAgg` primitive discussed in Section 2. As a first step, we identify for each pair of demands its least common ancestor (LCA), where we let $\text{LCA}(u, v)$ be the LCA of $u$ and $v$. This can be done in $O(n)$ work and $O(\log n)$ depth for each pair. Thus, we can compute the LCA for all demand pairs in $O(n + k \log n)$ work and $O(\log n)$ depth. As input to the second round, we maintain a variable $\mathsf{up}(w)$ for each node $w$ of the tree. Then, for every demand pair $\mathsf{dem}_{u,v}$, we add to both $\mathsf{up}(u)$ and $\mathsf{up}(v)$ the amount of $\mathsf{dem}_{u,v}$—and to $\mathsf{up}(\text{LCA}(u, v))$ the *negative* amount $-2\mathsf{dem}_{u,v}$. As such, the sum of all the values $\mathsf{up}$ inside a subtree rooted at $u$ is the amount of "upward" flow on the edge out of $u$ toward the root. This is also the net demand on this edge. Therefore, the demand $f_e$'s for all $e \in E(T)$ can be computed in $O(n + k \log n)$ work and $O(\log n)$ depth. Finally, mapping these back to $G$ and figuring out the cable type (i.e., computing $\min_i c_i\lceil \mathsf{dem}/u_i \rceil$) are straightforward and no more expensive than computing the all-pair shortest paths. Hence, we have the following theorem with the cost broken down by components:

**Theorem 5.1** *If the all-pairs shortest path problem on $G$ can be solved in $W_{APSP}$ work and $D_{APSP}$ depth and an FRT tree with no steiner node can be computed in $W_{FRT}$ work and $D_{FRT}$, then there is a randomized algorithm for the buy-at-bulk network design problem with $k$ demand pairs on an $n$-node graph that runs in $O(W_{APSP} + W_{FRT} + n + k \log n)$ work and $O(D_{APSP} + D_{FRT} + \log n)$ depth and produces an expected $O(\log n)$-approximation.*

This means that using the standard $O(n^3 \log n)$-work, $O(\log^2 n)$-depth algorithm for the all-pairs shortest path computation, and using our FRT algorithm, we have an expected $O(\log n)$-approximation for the buy-at-bulk network design problem that runs in $O(n^3 \log n)$ work and $O(\log^2 n)$ depth **whp.** If, on the other hand, the input already contains the all-pairs shortest paths distances, our algorithm runs in $O(n^2 \log n)$ work and and $O(\log^2 n)$ depth **whp.**

# 6. CONCLUSION

We gave an efficient parallel algorithm for tree embedding with $O(\log n)$ expected stretch. Our contribution is in making these bounds independent of the ratio of the smallest to largest distance, by recognizing an alternative view of the FRT algorithm and developing an efficient algorithm to exploit it. Using the embedding algorithms, we developed the first RNC $O(\log k)$-approximation algorithm for $k$-median and an RNC $O(\log n)$-approximation for buy-at-bulk network design.

## Acknowledgments

This work is partially supported by the National Science Foundation under grant numbers CCF-0964474, CCF-1016799, and CCF-1018188 and by generous gifts from IBM and Intel Labs Academic Research Office for Parallel Algorithms for Non-Numeric Computing. We thank the SPAA reviewers for their comments that helped improve this paper.

## References

[AA97] Baruch Awerbuch and Yossi Azar. Buy-at-bulk network design. In *Proceedings of the 38thFOCS*, pages 542–547, 1997.

[AGK+04] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for $k$-median and facility location problems. *SIAM J. Comput.*, 33(3):544–562, 2004.

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

[Bar96] Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *Proceedings of the 37thFOCS*, pages 184–193, 1996.

[Bar98] Yair Bartal. On approximating arbitrary metrics by tree metrics. In *Proceedings of the 30th ACM Symposium on the Theory of Computing (STOC)*, pages 161–168, 1998.

[BS11] Guy E. Blelloch and Julian Shun. A simple parallel cartesian tree algorithm and its application to suffix tree construction. In *ALENEX*, pages 48–58, 2011.

[BT10] Guy E. Blelloch and Kanat Tangwongsan. Parallel approximation algorithms for facility-location problems. In *SPAA*, pages 315–324, 2010.

[BV93] Omer Berkman and Uzi Vishkin. Recursive star-tree parallel data structure. *SIAM J. Comput.*, 22(2):221–242, 1993.

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

[JáJ92] Joseph JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.

[MP04] Ramgopal R. Mettu and C. Greg Plaxton. Optimal time bounds for approximate clustering. *Machine Learning*, 56(1-3):35–60, 2004.

[MRK88] Gary L. Miller, Vijaya Ramachandran, and Erich Kaltofen. Efficient parallel evaluation of straight-line code and arithmetic circuits. *SIAM J. Comput.*, 17(4):687–695, 1988.

[RR89] Sanguthevar Rajasekaran and John H. Reif. Optimal and sublogarithmic time randomized parallel sorting algorithms. *SIAM J. Comput.*, 18(3):594–607, 1989.

[Sei92] Raimund Seidel. Backwards analysis of randomized geometric algorithms. In *Trends in Discrete and Computational Geometry, volume 10 of Algorithms and Combinatorics*, pages 37–68. Springer-Verlag, 1992.

[Sha48] C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27, 1948.

[SV88] Baruch Schieber and Uzi Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM J. Comput.*, 17(6):1253–1262, 1988.

[Tam96] Arie Tamir. An $O(pn^2)$ algorithm for the $p$-median and related problems on tree graphs. *Operations Research Letters*, 19(2):59–64, 1996.

# APPENDIX

## A. VARIOUS PROOFS

PROOF OF LEMMA 3.12. Let $u \neq v \in X$ be given and let $y = \text{LCP}(\chi^{(u)}, \chi^{(v)})$. In $T''$ (also $T$ and $T'$), $y$ is the lowest common ancestor of $\chi^{(u)}$ and $\chi^{(v)}$. Let $i^* = |\text{LCP}(\chi^{(u)}, \chi^{(v)})|$. This means there is a vertex $w$ at distance at most $\beta \cdot 2^{\delta - i^* - 1}$ from both $u$ and $v$, so $d(u,v) \leq 2^{\delta - i^* + 1}$. Now let $a$ (resp. $b$) be the child of $y$ such that $T_a$ (resp. $T_b$) contains $\chi^{(u)}$ (resp. $\chi^{(v)}$). So then, we argue that $d_{T''}(\chi^{(u)}, \chi^{(v)}) \geq 2 \cdot 2^{\delta - i^*}$ because $\mathsf{label}(y)$ must differ from at least one of the labels $\mathsf{label}(a)$ and $\mathsf{label}(b)$—and such non-zero edges have length $2^{\delta - i^* + 1}$ since we doubled its length in Step (1). This establishes the stated bound. □

PROOF OF LEMMA 3.11. The depth of $T''$ is upperbounded by the length of the longest compressed partition sequence. Consider a vertex $v \in X$ and let $v_1, \ldots, v_{n-1} \in X$ be such that $d(v, v_1) > d(v, v_2) > \cdots > d(v, v_{n-1}) > 0$. The length of the compressed partition sequence of $v$ is upperbounded by the number of times the sequence $y_i = \min\{\pi(v_j) \mid n - 1 \geq j \geq i\}$ changes value. By Lemma 2.1, this is at most $O(\log n)$ **whp**. Taking union bounds gives the desired lemma. □

### A.1 Piecing Together the $k$-median Algorithm

#### A.1.1 Case I: $k \geq \log n$:

Let $Q = \{q_1, \ldots, q_K\} = \texttt{SuccSampling}(V, K)$, where $K = \max\{k, \log n\}$ and $\varphi : V \to Q$ be the mapping that sends each $v \in V$ to the closest point in $Q$ (breaking ties arbitrarily). Thus, $\varphi^{-1}(q_1), \varphi^{-1}(q_2), \ldots, \varphi^{-1}(q_K)$ form a collection of $K$ non-intersecting clusters that partition $V$. For $i = 1, \ldots, K$, we define $w(q_i) = |\varphi^{-1}(q_i)|$. We prove a lemma that relates a solution's cost in $Q$ to the cost in the original space $(V, d)$.

**Lemma A.1** *Let $A \subseteq Q \subseteq V$ be a set of $k$ centers satisfying*

$$\sum_{i=1}^{K} w(q_i) \cdot d(q_i, A) \quad \leq \quad \beta \cdot \min_{\substack{X \subseteq Q \\ |X| \leq k}} \sum_{i=1}^{K} w(q_i) \cdot d(q_i, X)$$

*for some $\beta \geq 1$. Then,*

$$\Phi(A) = \sum_{x \in V} d(x, A) \leq O(\beta \cdot c_{SS}) \cdot \Phi(OPT_k),$$

*where $OPT_k$, as defined earlier, is an optimal $k$-median solution on $V$.*

PROOF. For convenience, let

$$\lambda = \sum_{x \in V} d(x, Q) = \sum_{x \in V} d(x, \varphi(x)),$$

and so $\lambda \le c_{\text{SS}} \cdot \Phi(\text{OPT}_k)$. We establish the following:

$$\begin{aligned}
\sum_{x \in V} d(x, A) &\le \lambda + \sum_{i=1}^{K} w(q_i) \cdot d(q_i, A) \\
&\le \lambda + \beta \cdot \min_{\substack{X \subseteq Q \\ |X| \le k}} \sum_{i=1}^{K} w(q_i) \cdot d(q_i, X) \\
&\le \lambda + \beta \sum_{i=1}^{K} w(q_i) \cdot d(q_i, \varphi(\text{OPT}_k)) \\
&\le \lambda + 2\beta \sum_{i=1}^{K} w(q_i) \cdot d(q_i, \text{OPT}_k) \\
&\le \lambda + 2\beta \sum_{i=1}^{K} \sum_{x \in \varphi^{-1}(q_i)} \Big( d(q_i, x) + d(x, \text{OPT}_k) \Big) \\
&\le \lambda + 2\beta\lambda + 2\beta\Phi(\text{OPT}_k) \\
&= O(\beta \cdot c_{\text{SS}}) \cdot \Phi(\text{OPT}_k),
\end{aligned}$$

which proves the lemma. □

By this lemma, the fact that the $k$-median on tree gives a $O(\log |Q|)$-approximation, and the observation that $|Q| \le k^2$ (because $k \ge \log n$), we have that our approximation is $O(\log k)$.

### *A.1.2 Case II: $k < \log n$:*

We run successive sampling as before, but this time, we will use the parallel local-search algorithm [BT10] instead. On input consisting of $n$ points, the BT algorithm has $O(k^2 n^2 \log n)$ work and $O(\log^2 n)$ depth. Since $k < \log n$, we have $K = \log n$ and the successive sampling algorithm would give $|Q| \le \log^2 n$. This means we have an algorithm with total work $O(n \log n + k^2 \log^5 n)$ and depth $O(\log^2 n)$.

## B. TREE TRIMMING FOR K-MEDIAN

Another way to control the height of the tree is by taking advantage of a cruder solution (which can be computed inexpensively) to prune the tree. The first observation is that if $A \subseteq V$ is a $\rho$-approximation to $k$-center, then $A$ is a $\rho n$-approximation to $k$-median. Using a parallel $k$-center algorithm [BT10], we can find a 2-approximation to $k$-center in $O(n \log^2 n)$ work and $O(\log^2 n)$ depth. This means that we can compute a value $\beta$ such that if OPT is an optimal $k$-median solution, then $\Phi(\text{OPT}) \le \beta \le 2n \cdot \Phi(\text{OPT})$.

Following this observation, two things are immediate when we consider an *uncompacted* FRT tree:

1. If we are aiming for a $C \cdot \log n$ approximation, no clients could go to distance more than than $C \cdot \log n \cdot \Phi(\text{OPT}) \le C \cdot n\beta$. This shows twe can remove the top portion of the tree where the edge lengths are more than $Cn\beta$.
2. If a tree edge is shorter than $\tau = \frac{\beta}{8n^2}$, we could set its length to 0 without significantly affecting the solution's quality. These 0-length edges can be contracted together. Because an FRT tree as constructed in Theorem 3.1 is a 2-HST, it can be shown that if $T'$ is obtained from an FRT tree $T$ by the contraction process described, then for all $x \ne y$, $d_T(x, y) \le d_{T'}(x, y) + 4\tau$.

Both transformations can be performed on compacted trees in $O(n)$ work and $O(\log n)$ depth, and the resulting tree will have height at most $O(\log(8n^3)) = O(\log n)$. Furthermore, if $A \subseteq V$ is any $k$-median solution, then

$$\begin{aligned}
\Phi_d(A) \le \Phi_{d_T}(A) &= \sum_{x \in V} d_T(x, A) \\
&\le \sum_{x \in V} \big( d_{T'}(x, A) + 4t \big) \\
&\le \Phi_{d_{T'}}(A) + n \cdot \frac{\beta}{2n^2} \\
&\le \Phi_{d_{T'}}(A) + \Phi_d(\text{OPT}).
\end{aligned}$$