



Massively Parallel Tree Embeddings for High Dimensional Spaces

Amirmohsen Ahanchi
University of Maryland
College Park, Maryland, USA
ahanchi@cs.umd.edu

Alexandr Andoni
Columbia University
New York, New York, USA
ahanchi@cs.umd.edu

MohammadTaghi Hajiaghayi
University of Maryland
College Park, Maryland, USA
hajiagha@cs.umd.edu

Marina Knittel
University of Maryland
College Park, Maryland, USA
mknittel@cs.umd.edu

Peilin Zhong
Google Research
New York, New York, USA
pz2225@columbia.edu

ABSTRACT

Efficient computation on massive high-dimensional data greatly benefits from efficient embedding techniques into simpler metrics. Perhaps the most celebrated technique is the dimension reduction à-la Johnson and Lindenstrauss [46]. Another important method embeds the data into a tree metric space, first efficiently achieved by Bartal [14]. Both of these algorithmic tools are among the most general theorems with numerous applications.

In this paper, we study these two embedding methods in the Massively Parallel Computation (MPC) model. We develop a new *hybrid partitioning* algorithm which generalizes both random shifted grid and ball partitioning methods for generating tree embeddings. This leads to an $O(1)$ -round randomized MPC algorithm for embedding high-dimensional data into a tree while approximating the distance between any two points within a factor of $\tilde{O}(\log^{1.5} n)$ (and thus *distortion* $\tilde{O}(\log^{1.5} n)$) in expectation as long as the aspect ratio is $O(\text{poly}(n))$. This Euclidean result beats the lower bound of $\Omega(\log n)$ MPC rounds for tree embeddings of general metric spaces and can extend to a number of problems, including densest ball, minimum spanning tree, and Earth-Mover distance. Along the way, we implement and use Ailon and Chazelle's Fast Johnson Lindenstrauss Transform [2] with sublinear memory and $O(1)$ MPC rounds, which is of its own interest.

CCS CONCEPTS

• Theory of computation → Massively parallel algorithms.

KEYWORDS

Massively Parallel Computation; embeddings

ACM Reference Format:

Amirmohsen Ahanchi, Alexandr Andoni, MohammadTaghi Hajiaghayi, Marina Knittel, and Peilin Zhong. 2023. Massively Parallel Tree Embeddings for High Dimensional Spaces. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '23)*, June 17–19, 2023, Orlando, FL, USA. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3558481.3591096>



This work is licensed under a Creative Commons Attribution International 4.0 License.

SPAA '23, June 17–19, 2023, Orlando, FL, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9545-8/23/06.
<https://doi.org/10.1145/3558481.3591096>

1 INTRODUCTION

Massive data-driven computation benefits greatly from embedding finite metric spaces into simpler spaces. Specifically, high-dimensional massive datasets, while often highly practical, are frequently too large to store on commodity hardware. Therefore, there is much interest in finding efficient methods for transforming this data into low-dimensional spaces. For instance, one of the most famous algorithms in high-dimensional geometry is the Johnson-Lindenstrauss transform [46], which embeds n points in the Euclidean space with any dimension into the $O(\log n)$ -dimensional Euclidean space. Another branch of work solving this problem involves embedding metric spaces into tree metrics. A tree metric over n points is represented by an n -vertex tree, and therefore is also highly compact, requiring only $O(n)$ space. The main result of this paper is the first non-trivial massively parallel constant round extension of Bartal [14]'s famous probabilistic tree metric embeddings of geometric datasets. We additionally provide a space-efficient massively parallel adaptation of the Johnson-Lindenstrauss transform.

Rabinovich and Raz [52] showed that deterministically embedding a simple n -cycle into a tree metric requires $\Omega(n)$ distortion, or maximum proportional deviation between embedded and true distance. To circumvent this, Karp [48] leverages randomization to approximate a cycle by a path with low distortion. Alon, Karp, Peleg, and West [3] were the first to probabilistically embed arbitrary metric spaces into trees, however they required up to $2^{O(\sqrt{\log n \log \log n})}$ distortion to do so. Bartal's work greatly surpassed this, achieving an $O(\log^2 n)$ -approximation. A novel idea of Bartal's work in comparison with previous research is that it defines and utilizes *probabilistic partitions*, which ensures that two close points are more likely to be grouped together in the partition. By applying a hierarchy of probabilistic partitions, Bartal's algorithm embeds the input metric space into the so-called hierarchically well-separated tree (HST).

Tree embedding with HSTs has been improved a number of times since Bartal's inaugural work [15, 27, 49], culminating in the work of Fakcharoenphol, Rao, and Talwar [32], who improved the approximation factor to $O(\log n)$, notably yielding the first polylogarithmic approximation for the k -median problem. Since $\Omega(\log n)$ is also the lower bound [14], this result sets a good foundation for expanding tree embeddings in other directions [26, 39, 40].

Metric tree embeddings have already been studied in PRAM [8, 24, 34], a classic model of parallel computing. Given a general

metric space, Blleloch, Gupta, and Tangwongsan [24] designed a parallel $O(\log n)$ -approximate metric tree embedding algorithm using $O(n^2 \log n)$ work (i.e., number of operations) and $O(\log^2 n)$ depth (i.e., parallel time). Friedrichs and Lenzen [34] considered the shortest path metric given by a graph (i.e., graph metric) and gave a parallel $O(\text{poly}(\log n))$ -approximate metric tree embedding algorithm using $O(m + n^{1+\varepsilon})$ work and $O(\log n)$ depth where $\varepsilon > 0$ is an arbitrary constant and m is the number of edges of the input graph. Andoni, Stein, and Zhong [8] improved the work of [34] to $(m + n) \cdot \text{poly}(\log n)$ though with a larger distortion, a high degree $\text{poly}(\log n)$.

Due to the success of many modern massively parallel systems such as MapReduce [30], Hadoop [55], and Spark [57], a more refined model of parallel computing emerged — Massively Parallel Computation (MPC) [18, 38, 47]— and has led to the development of new parallel algorithms in recent years. In this model, data is distributed to multiple machines where each machine has a sublinear amount of memory. We alternate between rounds of computation and rounds of communication where each machine can only send messages with size bounded by its local memory in a single round. Since communication is always the bottleneck of the model, the goal in MPC is to design an algorithm with few rounds (parallel time). We know that t -depth PRAM algorithms can be simulated in MPC in $O(t)$ rounds [54]. Thus, in MPC, the simulation of any above mentioned PRAM algorithm would require $\Omega(\log n)$ parallel time. On the other hand, an $o(\log n)$ -round MPC algorithm is always more desired in practice and faster MPC algorithms exist for many problems (see e.g., [6, 7, 29, 37]).

Thus emerges the following natural question that we study in this paper:

Can we design an $o(\log n)$ -round MPC algorithm for metric tree embedding?

The answer is no for general input metric spaces (e.g., the graph metric) with polylogarithmic distortion unless the 1-vs-2Cycle Conjecture [56] is false. In geometric space, Arora [9]’s grid partitioning solves this in $O(1)$ MPC rounds with $O(\log^2 n)$ distortion.

We are the first to break this distortion barrier. On n points in $\mathbb{R}^d$ with aspect ratio $\text{poly}(n)$, there exists an $O(1)$ -round MPC algorithm for $\tilde{O}(\log^{1.5} n)$ -approximate metric tree embedding.¹² This yields $O(1)$ -round $\tilde{O}(\log^{1.5} n)$ -approximate MPC algorithms for Euclidean: minimum spanning tree, Earth-Mover distance, and densest ball.

We propose a new hierarchical probabilistic partitioning method to embed data in $\mathbb{R}^d$ into a tree with distortion $\tilde{O}(\log^{1.5} n)$ using constant MPC rounds and low memory. Generally speaking, these methods iteratively partition the data and then recurse on each part in the partition until singletons or empty sets are reached. This yields a tree whose edges we weight and whose leaf set is the dataset. Its tree metric defines pairwise embedded distances. Our algorithm, *hybrid partitioning*, can be seen as a generalization of two existing partitioning methods: Arora [9]’s grid partitioning and Charikar et al. [27]’s ball partitioning.

¹The aspect ratio of a point set is the ratio between the largest and the smallest interpoint distance.

² $\tilde{O}(f(n))$ denotes $O(f(n) \cdot \text{polylog}(f(n)))$.

The main novelty of our methods is a hybridization of the two methods at each level of partitioning. Specifically, to partition the data, we group dimensions into r buckets, executing a ball partitioning on each bucket, and combining them with grid partitioning-like methods. If we set $r = 1$, all dimensions are in one bucket so the algorithm simply ball partitions the data. If $r = d$, the ball partitioning step simplifies greatly, and we end up effectively grid partitioning all points.

Our algorithm illustrates the trade-offs between the two methods in the parallel setting: grid partitioning methods reduce local memory and ball partitioning methods improve distortion. It turns out the key in our methods is to guarantee that an entire partition of the data can be stored in local memory. This becomes complicated using ball partitioning, since it requires a large number of attempts (and therefore, entire grids to store) to encode a partition. Our hybridization finds a nice way to reduce this space by only running ball partitions on subsets of dimensions.

Even with our space-reducing hybridization, a preprocessing application of the Johnson-Lindenstrauss transform to the input data is required to reduce dimensionality. Therefore, we include, as a result of independent and dependent interest, an efficient MPC implementation of the Fast Johnson-Lindenstrauss transform (Theorem 3). It achieves an $O(\log n)$ -distortion embedding in $O(1)$ MPC rounds with low memory. The use of the fast transform over the original in particular allows for an important reduction in the total space for high-dimensional data.

1.1 Massively Parallel Computation

We work in the *Massively Parallel Computation* (MPC) model [18, 47]. MPC is an abstraction of MapReduce [30] that models programming frameworks such as Hadoop [33], Spark [57], and Flume [25]. MapReduce is used across industry, and is known for its fault tolerance and compatibility with commodity hardware. On graphs specifically, MPC has been used in many applications such as clustering [16, 42, 56] and Earth-Mover distance [5], as well as theoretical problems like connectivity [6, 7, 12, 21], matching and vertex cover [1, 10, 23, 35], minimum spanning tree [5], and coloring [11, 19]. Recent research has also explored adaptations of MPC [20, 22, 43, 44, 54]. MPC is highly practical and for this reason, we study it in this work.

In MPC, the input is distributed across multiple machines. The computation proceeds in rounds, wherein each machine executes a local polynomial-time computation. At the end of the round, machines may send messages to and receive messages from any other machines. The total size of messages sent or received by a machine in a round is bounded by its local memory. MPC algorithm efficiency is measured by: the number of rounds (parallel time), the local memory, and the total space (number of machines times the local memory).

In this work, we consider MPC algorithms in the geometric context, where the input data contains n points in $\mathbb{R}^d$, represented by d -dimensional vectors. We use the most restrictive version of MPC where local space per machine is $O((nd)^\varepsilon)$ for any constant $\varepsilon \in (0, 1)$ —termed the “fully scalable” regime [6, 7]. All our algorithms are fully scalable, take $O(1)$ rounds, and use total space near linear in the input size $n \cdot d$.

1.2 Grid Partitioning Methods for Tree Metrics

We now describe two grid partitioning methods that we will extend in our work: random shifted grids and ball partitioning.

1.2.1 Random Shifted Grids. The first is the standard random shifted grid introduced by Arora [9]. Consider a geometric space in d dimensions. A random shifted grid is just a standard grid with cell width w whose origin is translated by some vector $(x_1, \dots, x_d)$ where x_i is drawn uniformly at random from $[0, w]$. Equivalently, each cell is translated by the vector $(x_1, \dots, x_d)$. A visualization can be seen in Figure 1a.

Definition 1. *Given a cell width parameter w , consider a grid G of cell length w shifted randomly by a vector sampled uniformly at random from $[0, w]^d$. Place each point p into a partition representing the cell that contains it. This partitioning is a **random shifted grid partitioning** with scale w .*

We now discuss how to create a hierarchical partitioning from these random shifted grids. At this point, we will often refer to a *level*, which refers to a flat partitioning in a hierarchy, or alternatively, the recursive level in the hierarchical partitioning algorithm, starting with zero at the top. Let Δ be the aspect ratio (the maximum ratio between the maximum and minimum pairwise distances), $\mathcal{B}$ be a bounding box over our data (which we can say has width Δ), and ℓ be a parameter defining how many cells our grid should have, and how much it should increase at each level. We start by sampling a random shifted grid over $\mathcal{B}$ with cell width $w = \Delta/\ell$. Each point then falls into a cell in the grid. We create a partitioning of data where each partition corresponds to a non-empty cell such that it contains all points contained within the cell. We then recurse to make a hierarchical structure. The idea is to create a more refined grid (i.e., a grid with a smaller cell width) at each consecutive step. Generally, at the i th level in the hierarchy, partition each partition in the previous level using a randomly shifted grid of width Δ/ℓ^i . This then yields a new partitioning over our data, where partitions are more numerous and smaller, which we add to the hierarchy. For any partition, we stop partitioning as soon as it has one or no points.

The hierarchy defined by the random shifted grid partitioning procedure can be simply viewed as a tree. Let $\mathcal{B}$ be the root vertex. Then for each cell we create, add a vertex to the tree with parent vertex corresponding to the cell's parent cell (i.e., the one which contains a superset of its points). Clearly this is a tree, and the leaves will either be empty (in that case, we can simply not create such a node) or they will represent a single datapoint. Therefore, we have created a tree structure to represent the data. Consider labeling each tree edge with weight $w\sqrt{d}$, where w was the cell width on that level. Then, the distance between two points is defined as the weight of the shortest path between the two points.

Grid partitioning is a nice, simple, classic technique that has inspired many results, including ball partitioning and our hybrid partitioning. It would be nice to simply use grid partitioning out-of-the-box, and it is not too hard to see that this can be implemented efficiently in MPC in $O(1)$ rounds with no significant local and total space issues. However, grid partitionings only achieve an $O(\log^2 n)$ distortion. We can do better.

1.2.2 Ball Partitioning. The ball partitioning method, depicted in Figure 1b, was introduced by [27] for the purpose of derandomizing Bartal's algorithm. In spirit, it works quite similarly to random shifted grids, however we create partitions based off a grid of balls instead of the cells in a grid. In this method, we have two width parameters: the cell width and the ball radius. For simplicity of understanding, say that the ball radius is w and the cell width is $\ell = 4w$.

To create a single partitioning, first sample a random shifted grid of cell width $4w$ over the space. At each grid intersection point in our bounding box, create a ball of radius w . Note here that it is necessary that the cell width is more than twice as large as the ball radius, otherwise the balls will overlap and a point may fall into two balls. Even if the balls do not overlap, the resulting partitioning will not necessarily partition the grid entirely, as some points may fall outside of all balls. To account for this, we simply continue to sample random shifted grids and create partitions for each grid, removing covered points as we go. We do this until all points are covered (or stop at some point and know that we succeed at covering all points with some probability).

Definition 2. *Given a cell length parameter ℓ and radius w with $w = \frac{1}{4}\ell$, consider a sequence of grids $G_1, G_2, \dots$ of cell length ℓ shifted randomly by vectors $s_1, s_2, \dots$ sampled uniformly at random from $[0, \ell]^d$. Place a ball of radius w at each grid point for all $G_1, G_2, \dots$. Place each point p into the first ball that contains it according to the grid ordering. This partitioning is a **ball partitioning** with scale w (or scale ℓ).*

The described method defines a partitioning at a single layer in the hierarchy. To create an entire hierarchy, we use the exact same strategy employed by the random shifted grids method, but instead creating our partitionings at each level using the ball partition.

Ball partitionings, while slightly more complicated, achieve a much nicer $O(\log^{1.5} n)$ distortion. The issue with this method is that it requires too much space to implement efficiently in MPC. Namely, we need to generate a large number of grids in order to cover the entire space, which will be exponential in d . Even though we reduce d to $O(\log n)$ using the Johnson-Lindenstrauss transform, this dependency is still too large. We later show in Lemma 7 how to reduce this dependency by adding more buckets of dimensions.

1.3 Our Contributions

We propose fully scalable, constant-round MPC algorithms for embeddings of geometric data in the MPC model. The set of points $P \subset \mathbb{R}^d$ is encoded as a set of d -dimensional vectors (and therefore requires $O(nd)$ total space) and is assumed to have a bounded aspect ratio. Without loss of generality, we regard the coordinates of points as integers from $[\Delta] = \{1, 2, \dots, \Delta\}$. For two points $x, y \in \mathbb{R}^d$, we use $\|x - y\|_2$ to denote their Euclidean distance. Our goal is to output a weighted tree containing all points in P such that $\text{dist}_T(p, q)$, the total length of the path from p to q on T (i.e., the tree metric on T), is close to $\|p - q\|_2$. Note that since the input size is $O(nd)$, $O((nd)^\epsilon)$ local space is considered fully scalable.

Our main result is the first fully scalable constant round MPC algorithm to break the $O(\log^2 n)$ expected distortion (i.e., the multiplicative deviation of $\mathbb{E}_T[\text{dist}_T(p, q)]$ from $\|p - q\|_2$) implied by Arora [9]'s grid partitioning. To our knowledge, other than the

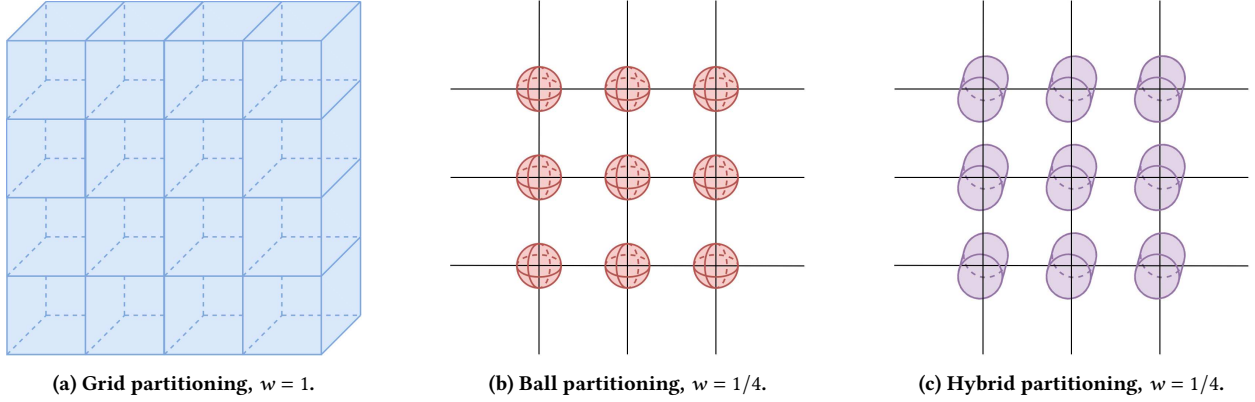


Figure 1: We depict one level (and one sample) of each discussed partitioning method. In grid partitioning (1a), we partition the grid into hypercubic cells of width 1 shifted by a random vector. In ball partitioning (1b), we place a ball of radius $1/4$ at each intersection of grid boundaries. Note that one instance of this placement is not sufficient to partition an entire space, as some parts are not covered by balls. Thus, we need to repeatedly draw randomly shifted grids and place balls at the intersections until every point in the space is covered by a ball. In hybrid partitioning (1c) with $r = 2$, we run a ball partitioning with ball radius $1/4$ on buckets of dimensions. If the z axis is sticking out towards the reader, then this involves two buckets: $\{x, y\}$ and $\{z\}$. We do a ball partitioning of the points projected onto the xy -plane and the z -axis independently, and then intersect them to get a partitioning. Since partitions in the xy plane are circles and partitions on the z axis are intervals, taking their intersection in 3-dimensional space results in cylindrical-shaped partitions.

work of Arora, this is the only constant-round MPC algorithm for tree embeddings of high-dimensional data. Note that the success probability $1 - 1/\text{poly}(n)$ holds for any polynomial function in n .

Theorem 1. Consider a set of n points $P \subseteq [\Delta]^d$ for $\Delta \in \mathbb{Z}_{\geq 1}$. There is an $O(1)$ -round randomized MPC algorithm which computes a weighted spanning tree T over P when it succeeds, such that $\forall p, q \in P$,

- (1) $\text{dist}_T(p, q) \geq \|p - q\|_2$.
- (2) $\mathbb{E}_T [\text{dist}_T(p, q)] \leq O(\sqrt{\log n} \cdot \log \Delta \cdot \sqrt{\log \log n}) \cdot \|p - q\|_2$.

The success probability is at least $1 - 1/\text{poly}(n)$. The algorithm uses $O(n \cdot d + n \log n \cdot (\log \Delta \cdot \log \log n + \min(d, \log^2 n)))$ total space and each machine holds $O((nd)^\epsilon)$ local space for an arbitrary constant $\epsilon \in (0, 1)$. If the algorithm fails, it reports failure.

The algorithm that achieves this result has two parts. The first is an efficient implementation of the *Fast Johnson Lindenstrauss transform*, a famous technique that reduces any high dimensional space into at most $O(\log n)$ dimensions. The second is a novel hybrid partitioning algorithm which combines Arora’s random shifted grid partitioning [9] and Charikar et al.’s ball partitioning [27] methods. On $O(\log n)$ -dimensional data, this can be efficiently implemented in MPC. Together, these yield our main result.

This result stands in contrast to the results for general shortest-path metric of graphs. Conditioning on the 1-vs-2Cycle Conjecture [56] (which postulates that distinguishing between a graph that is one n -cycle or two disjoint $n/2$ -cycles requires $\Omega(\log n)$ MPC rounds), any fully scalable MPC algorithm for *general* graph connectivity needs $\Omega(\log n)$ rounds. If a graph is disconnected, then there are some $p, q \in P$ that are infinitely far apart. Any multiplicative approximation of the shortest path distance between p and q by a fully scalable MPC algorithm would approximate that distance as infinite, thus identifying the graph as disconnected.

It therefore requires $\Omega(\log n)$ rounds. This means that there is no multiplicative-approximate $o(\log n)$ -round fully scalable MPC metric tree embedding algorithm for the graph metric under the 1-vs-2Cycle Conjecture [56]. While we do not break this important barrier, our results show that the 1-vs-2Cycle Conjecture implies an infinite approximation gap for sublogarithmic MPC round shortest path distance in metric and geometric graphs.

1.3.1 Methods: Hybrid Partitioning. To achieve our result, we introduce the notion of *hybrid partitioning*, which combines two different geometric partitioning methods. Both partitioning methods are illustrated in Figure 1. The first is the standard random shifted grid introduced by Arora [9], where the data is partitioned by the cells of a grid, and the origin of the grid is offset by a random vector. The second is the randomized ball partitioning method, where the same random grid is used but instead of partitioning into the grid cells, balls of radius $1/4$ the width of the cells are placed at each line intersection [27]. These define partitions. This is repeated until each point is covered.

The goal of a hybrid partitioning is to create an intermediate method which combines strategies from both partitioning algorithms. When parameterized to one extreme, hybrid partitioning is equivalent to grid partitioning. At the other extreme, it is equivalent to ball partitioning. We define hybrid partitioning with parameters $w, \ell \leq \Delta$ and $r \leq d$, where ℓ and w determine the scale of the partitions (similarly to ball partitioning) and r controls how to hybridize grid and ball partitioning. The following formal definition defines a flat partitioning of the space (and the data). This can be made hierarchical by recursing on each partition. The resulting hierarchy is represented by a weighted tree: our embedding.

Definition 3. In a d -dimensional space, consider bucketing all d dimensions into r buckets $\{\{1, \dots, d/r\}, \{d/r + 1, \dots, 2d/r\}, \dots, \{d -$

$d/r+1, \dots, d\}$ for parameter $r \leq d$. Let ℓ be a scaling parameter and $w = \frac{1}{4}\ell$.³ For an arbitrary point $p \in \mathbb{R}^d$, let $p^{(i)} \in \mathbb{R}^{d/r}$ be obtained from restricting (projecting) p on the dimensions in bucket i . For each bucket $1 \leq i \leq r$, run a ball partitioning on $P^{(i)} = \{p^{(i)} : p \in P\}$ with parameters w and ℓ . If a partitioning of $\mathbb{R}^d$ satisfies that p and q are in the same partition if and only if they are in the same partition for all buckets, we call it an r -**hybrid partitioning** with **scale** w (or **scale** ℓ).

An illustration of hybrid partitioning on $\mathbb{R}^3$ can be seen in Figure 1c. Abstracting away the specific functionality of the algorithm, we can see the similarities between ball and grid partitioning, and how hybrid partitioning is an intermediate strategy. In this example, $r = 2$. If $r = 3$, hybrid partitioning must partition the space into cubes. If $r = 1$, it must partition the space into spheres.

We start with a sequential algorithm which is described in Section 3. Later, in Section 4, we will show how this algorithm can be implemented fully scalably in the MPC model, which results in our Theorem 1. We show that the sequential algorithm achieves the following guarantees:

Theorem 2. Consider a set of n points $P \subseteq [\Delta]^d$ for $\Delta \in \mathbb{Z}_{\geq 1}$ and a parameter $r \in [d]$. Algorithm 1 computes a weighted spanning tree T over P such that $\forall p, q \in P$, $\|p - q\|_2 \leq \text{dist}_T(p, q)$ and $E_T[\text{dist}_T(p, q)] \leq O(\sqrt{d} \cdot r \cdot \log \Delta) \cdot \|p - q\|_2$.

We now describe our sequential algorithm for hybrid partitioning. Without loss of generality, we suppose r divides d . We start by grouping the dimensions into r buckets each containing d/r dimensions. For each bucket, we project the data points into the space defined by these dimensions and then we run a ball partitioning (see Section 1.2.2 for a detailed description) with scale parameter $\Theta(w)$, meaning that the ball radius is w . Then each point is associated with one partition for each bucket. To join the buckets, we simply take the intersection of partitions over all buckets. For instance, two points p and q are in the same final partition under the hybrid partitioning if they are in the same partition obtained by the ball partitioning for every bucket.

To compute a tree embedding, we iteratively call the hybrid partitioning. At the beginning, $w = \Delta/2$. Once we obtain the partitions from the hybrid partitioning, we reduce the scale parameter w by a factor of 2 and recursively apply the hybrid partitioning on each partition. This yields a hierarchy whose root is the partition of all data points, and leaves represent a partitioning into singletons. As we move down the hierarchy, we connect each child node to its parent node with a weight proportional to $w \cdot \sqrt{r}$, where w is the scale parameter of the recursive *level*, and we show that $w \cdot \sqrt{r}$ is an upper bound of the diameter of a partition in the current level. Finally, we obtain a weighted tree. We refer readers to Section 3 for more details.

To see why this algorithm generalizes grid and ball partitioning, we consider extreme values of r . When $r = 1$, we have a single bucket. Clearly, then, our partitioning algorithm just executes the ball partitioning algorithm. When $r = d$, we partition each dimension separately. Therefore, the shapes of the intersections of all partitions end up being hypercubes. In our implementation, we

let the ball radius be w and the cell length be $4w$, which means there is space between the hypercubes, unlike grid partitioning. However, if we instead let the ball radius be $w/2$, we eliminate all the uncovered space for $r = d$, and our algorithm becomes equivalent to grid partitioning. Therefore, this is a hybridized version of the two partitioning methods. This algorithm also, notably, is implementable in the MPC model.

An interesting property of our hybrid partitioning is that we can find a bound on the cutting probability of two points (i.e., the probability that two points are assigned to different partitions) that is independent of r . This means that we can ensure some probability that two points are not separated at some level in the hierarchy regardless of the selection of r . Our bound for the diameter of partitions, however, is dependent on r . These two bounds are as follows. Note that the separation probability bound is only useful when $w > \sqrt{d}\|p - q\|_2$. Since w starts as Δ , this is true for at least one pair of points (though in most cases, many pairs) in the initial stages of the algorithm.

Lemma 1. Consider a hybrid partitioning with parameters $w \in \mathbb{R}_{>0}$ and $r \leq d$ in the d -dimensional Euclidean space. For any two points $p, q \in \mathbb{R}^d$, the probability that p and q are assigned to different partitions is at most $O\left(\sqrt{d} \cdot \frac{\|p - q\|_2}{w}\right)$. If $p, q \in \mathbb{R}^d$ are assigned to the same partition, $\|p - q\|_2 \leq O(\sqrt{r} \cdot w)$.

1.3.2 Methods: Johnson Lindenstrauss in MPC. In addition to our metric tree embedding result, we devise an efficient MPC implementation of the Fast Johnson-Lindenstrauss Transform (FJLT) [2]. The FJLT utilizes sparse projections and the randomized Fourier transform to improve upon the standard Johnson-Lindenstrauss transform [46]. It reduces the dimension of data points to $O(\log n)$ with distortion at most $(1 \pm \xi)$ for any $\xi > 0$. The guarantee of our algorithm is the following, beating previous work in terms of total space by a factor of $\log n/\xi^2$:

Theorem 3. Consider a set of points $P = \{p_1, p_2, \dots, p_n\} \subset \mathbb{R}^d$. Let $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^k$ be Fast Johnson Lindenstrauss Transform with $k = \Theta(\xi^{-2} \log n)$ for $\xi \in (0, 0.5)$. There is an MPC algorithm which outputs $\phi(p_1), \phi(p_2), \dots, \phi(p_n)$ in $O(1)$ rounds. In addition, the total space of the algorithm is at most $O(nd + \xi^{-2}n \log^3 n)$ and each machine holds $O((nd)^\epsilon)$ local space for an arbitrary constant $\epsilon \in (0, 1)$.

Details on the implementation and proofs can be found in Section 5. Along with being a separate interesting result in its own right, the ability to execute the transform in MPC is necessary for our main result. If we were to omit the transform as the first step before the hybrid partitioning, the hybrid partitioning would work on a potentially n -dimensional dataset. However, as we will see in the analysis for the MPC implementation of hybrid partitioning in Section 4, this would require an intractable computation. Specifically, the number of “balls” (i.e., partitions) at each partitioning is required to be exponential in d in order to cover the whole space with high probability. Since we must store the entire partitioning, potentially, this would make the total space exponential in n , which is quite excessive in MPC. However, we show that the dimensionality reduction of the Johnson Lindenstrauss transform is sufficient to yield an efficient MPC implementation.

We briefly note that the fast transform, as opposed to the original method, yields a more efficient MPC algorithm on high-dimensional

³Without loss of generality, we assume d is divisible by r . Otherwise, we can concatenate 0s to each point to increase d to d' by a factor at most 2 and $d' \bmod r = 0$.

data. Specifically, the standard transform would require $O(nd \log n)$ total space. However, if $d = \omega(\log^2 n)$, the rest of our algorithm (along with the fast transformation) achieves about $O(nd)$ total space. Therefore, we are able to achieve a total space reduction that is proportional to $\log n$ using the fast transform. In large-scale models such as MPC, even careful improvements like this can yield significant gains.

1.3.3 Applications. Both our metric tree embedding and fast Johnson Lindenstrauss results are useful for creating compact representations of high-dimensional geometric spaces. In fact, Theorem 1 is obtained by running the result from Theorem 3 to reduce the dimension of the data with constant distortion and then an MPC implementation of Algorithm 1 with $r = O(\log \log n)$ (Theorem 2).

Metric tree embeddings can be used to solve or approximate many problems in graph theory, and our results are no exception. This work can be extended to numerous applications, and we note three important ones below. Proofs can be found in the appendix. Note that an (α, β) -approximate densest ball is a bicriteria solution indicating that given a target diameter D , we approximate the problem of finding the ball that contains the most points within a factor of α with up to a β -multiplicative violation of the ball diameter. In other words, the ball may have diameter up to βD .

Corollary 1. *Consider a set of n points $P \subseteq \mathbb{R}^d$ with aspect ratio $\Delta \in \mathbb{Z}_{\geq 1}$. There is an $O(1)$ -round randomized MPC algorithm which computes (on P with probability at least $1 - 1/\log \log n$) a:*

- (1) $(1 - O(1/\log \log n), \tilde{O}(\log^{1.5} n))$ -approximate densest ball
- (2) $\tilde{O}(\log^{1.5} n)$ -approximate minimum spanning tree
- (3) $\tilde{O}(\log^{1.5} n)$ -approximate Earth-Mover distance

It uses $O(n \cdot d + n \log n \cdot (\log \Delta \cdot \log \log n + \min(d, \log^2 n)))$ total space and each machine holds $O((nd)^\epsilon)$ local space for an arbitrary constant $\epsilon \in (0, 1)$.

There are a few important things to note regarding these applications. First, to our knowledge, densest ball has not been studied in MPC and therefore our result is the first in this area. It is highly related to the problem of finding the densest subgraph of a given graph. In this problem the idea is to identify a subgraph H of an unweighted graph G that minimizes the density $d(H) = |E(H)|/|H|$, where $E(H)$ is the set of edges with both endpoints in H . While this has been studied in the sublinear regime [13, 36], it does not imply any results for densest ball.

Additionally, a recent work [28] proposed an efficient $\tilde{\Theta}(\log n)$ -approximate algorithm for EMD and MST in $\mathbb{R}^d$. However, their work is less broad since they directly compute EMD and MST, whereas we provide a general low-distortion embedding algorithm that can be used to solve a wide range of problems. In addition, there are applications where maintaining a space-efficient embedding of a dataset before computation may be highly practical. Therefore, our result is still of unique interest.

Finally, it is also notable that storing data on trees provides a unique structure for data computation. For instance, related works [17, 44] introduced efficient low-memory MPC and AMPC algorithms for solving dynamic programs on trees.⁴ Consider a problem that

⁴Adaptive MPC, a related model where machines have adaptive in-round read-only access to a distributed hash table.

can be formulated on a tree embedding (i.e., where leaves correspond to the data set) with distortion α such that the problem can be approximated within a factor of $f(\alpha)$. Then we can apply these algorithms on top of our embedding to achieve an $f(\tilde{O}(\log^{1.5}(n)))$ -approximation. Since the AMPC algorithm of [44] runs in constant rounds, then this process would require $O(1)$ rounds overall to embed and compute. Unfortunately, the MPC algorithm of [17] requires $O(\log n)$ rounds, and therefore does not fully leverage our constant round complexity. Future work in this area may reveal more interesting results.

1.3.4 Related Lower Bounds. When the aspect ratio is $\text{poly}(n)$, the distortion of our current metric tree embedding is $\tilde{O}(\log^{1.5} n)$. One of the future directions is to improve this approximation ratio in the MPC model. A natural goal would be to improve the distortion to $O(\log n)$ since any $o(\log n)$ -distortion metric tree embedding would imply an embedding of the Earth-Mover distance into ℓ_1 with distortion better than the long-standing state-of-the-art embeddings [51] (even for planar Earth-Mover distance).

A number of related problems also exhibit similar apparent limitations. Embedding an n -point metric in ℓ_2 space into probabilistic trees needs at least $\Omega(\sqrt{\log n})$ distortion. This follows from a result of Rao [53] which states any finite planar metric of cardinality n , in particular a $\log n$ -level diamond graph, can be embedded into ℓ_2 space with distortion $O(\sqrt{\log n})$ (which is tight according to Lee and Naor [50]) and another result of Gupta, Newman, Rabinovich, and Sinclair [41] which states a $\log n$ -level diamond graph needs distortion of at least $\Omega(\log n)$ to probabilistically embed into trees. Therefore, the distortion of embedding Euclidean points into trees must be at least $\Omega(\sqrt{\log n})$. While some of the above discussion does not enforce bounds on our problem, they are indicative of the difficulty of metric tree embedding in general. It is an open question to further explore this gap for high-dimensional Euclidean spaces.

2 PRELIMINARIES

Here we introduce some preliminary definitions that will be useful in describing our algorithm and our results. This work provides tree embeddings on geometric data that exhibit two properties: they *dominate* the original geometric space and have small *distortion*. A metric space dominates another if the distance between any pairs of points is not smaller in the new metric space. This is a baseline assumption that many embeddings are shown to satisfy, as in Fakcharoenphol, Rao, and Talwar [32]. They define domination as follows.

Definition 4 ([32]). *A metric space $\mathcal{M}$ dominates another metric space $\mathcal{N}$ for all $q, p \in P$ for some set of points P if $d_{\mathcal{M}}(q, p) \geq d_{\mathcal{N}}(q, p)$, where $d_{\mathcal{M}}$ and $d_{\mathcal{N}}$ represent the distance function in each metric space respectively.*

Distortion (in our case, bi-Lipschitz distortion) measures the difference between the distance between two points in the original space and two points in a tree embedding. It is a measure of the goodness of the embedding in that low distortion implies we better approximate all pairwise distances, as measured by the largest proportional deviation.

Definition 5 ([24]). *A metric space $\mathcal{M}$ has α distortion over another metric space $\mathcal{N}$ if and only if $\mathcal{M}$ dominates $\mathcal{N}$ and for any points*

x and y , $d_M(x, y) \leq \alpha d_N(x, y)$, where d_M and d_N represent the distance function in each metric space respectively.

If M is generated by a random process or represents a distribution over metric spaces, we instead view distortions in expectation, where $E[d_M(x, y)] \leq \alpha d_N(x, y)$. Specifically, our algorithm is randomized, so we will use this probabilistic interpretation of distortion. In this paper, we develop algorithms that dominate the original geometric space with low distortion.

3 HYBRID PARTITIONING AND ITS DISTORTION

Our sequential hybrid partitioning algorithm, Algorithm 1 (subroutines BuildGrids and BallPart can be found in the appendix, is a generalization of both Arora [9]’s random shifted grid and Charikar et al. [27]’s random ball partition method. As we discussed in Section 1.2, the reason we cannot use either of these independently for a massively parallel algorithm is actually quite simple. For random shifted grids, the problem is that the resulting distortion is $O(\log^2 n)$. Our methods strive to achieve a better approximation factor than this. For ball partitions, the local space required is exponential in the dimension of the problem since we require many random shifted grids to cover all points in P . Even for d logarithmic in n , this method is unattainable in MPC.

Algorithm 1 Hybrid Partitioning: A Sequential Tree Embedding Algorithm

Input: Point set $P \subseteq [\Delta]^d$ and parameters $r \in [d]$, the number of buckets, and $U \in \mathbb{N}$, the number of grids

Output: T , a tree embedding of P

```

// Bucket the dimensions  $[d]$  into  $r$  buckets
for  $j \in [r]$  do
   $j_0 \leftarrow (d/r) \cdot (j - 1)$ 
   $P^{(j)} \leftarrow \{p^{(j)} \mid p \in P : p^{(j)} = (p_{j_0+1}, p_{j_0+2}, \dots, p_{j_0+d/r})\}$ 

// Create a full ball partitioning and a corresponding hierarchy
for each bucket
  For all  $j \in [r]$ :  $G^{(j)} = \text{BuildGrids}(P^{(j)}, r, U)$ 
  For all  $j \in [r]$ :  $T_j \leftarrow \text{BallPart}(P^{(j)}, G^{(r)})$ 
  If any ball partitionings failed, halt and report failure

// Join the partitionings to make a single, unified hierarchy
For all  $j \in [r]$ :  $v_{0,j} \leftarrow$  the root of  $T_j$ 
 $v_0 = (v_{0,1}, v_{0,2}, \dots, v_{0,r})$  // A vertex for the single cluster
containing all of  $P$ 
 $T \leftarrow (\{v_0\}, \emptyset)$ 
Let  $C : T \rightarrow 2^P$  where  $C(v_0) = P$  // Identifying the cluster
corresponding to a vertex
while  $\exists v = (v_1, v_2, \dots, v_r) \in \text{leaves}(T)$  such that  $|C(v)| > 1$  do
   $S = \text{children}(v_1) \times \text{children}(v_2) \times \dots \times \text{children}(v_r)$ 
  for  $u = (u_1, u_2, \dots, u_r) \in S$  do
     $P_u \leftarrow \{p \in P : \forall j \in [r], p^{(j)} \text{ is in the cluster corresponding to } u_j\}$ 
    If  $P_u \neq \emptyset$ , add  $u$  to  $T$  as a child of  $v$  and set  $C(u) = P_u$ .
return  $T$ 

```

The advantage of ball partitioning, however, is that it achieves a lower distortion. Therefore, the main idea of the hybrid partitioning method is to combine these two methods such that it is implementable in MPC, like random shifted grids, while still obtaining the improved distortion from ball partitioning. To do this, we introduce the notion of bucketing. To see how bucketing works, consider our bounding box $\mathcal{B}$ of our data, whose width is Δ , the aspect ratio of the data. To partition this bounding box, we start by bucketing the d dimensions into r buckets. That is, for a vector $\vec{x} = (x_1, \dots, x_d)$, let it be bucketed as follows:

$$x^{(1)} = (x_1, \dots, x_{d/r}), x^{(2)} = (x_{d/r+1}, \dots, x_{2d/r}), \\ \dots, x^{(r)} = (x_{d-d/r+1}, \dots, x_d)$$

In a sense, the bucketing is taking a single point in space and projecting it into a number of different orthogonal subspaces. Since all dimensions are contained in exactly one subspace each, we do not lose any information in this process. This describes how one point is broken up into different projections. Over a set P of points, we create sets for each bucket of dimensions. The set contains the projection of each point in P into the respective bucket.

$$P^{(1)} = \{x^{(1)} \mid \forall x \in P\}, P^{(2)} = \{x^{(2)} \mid \forall x \in P\}, \\ \dots, P^{(r)} = \{x^{(r)} \mid \forall x \in P\}$$

At a high level, our algorithm, Algorithm 1, will create a ball partitioning on each bucket with cell width $4w$ and ball radius w for each level with different scale parameter w , and only group two points together if they are in the same partition for each bucket. Specifically, for each $i \in [r]$, let C_i be the partitioning created by the ball partitioning on only the dimensions in bucket i (i.e., the $(i-1)d/r+1$ through id/r dimensions of each data point). We create our C partitioning as follows: for each point p , to find the other points in its partition, let C^p be the set of partitions in $C_1, \dots, C_r$ that contain p (note that some may be empty, as a ball partitioning may not cover p). Take the intersection of all partitions in C^p to form the partition C^p ($C^p = \bigcap_{C \in C^p} C$). If C^p contains any other point, then it will be a partition in our new partitioning. Like in the ball partitioning method, we repeat this until all points are covered. If at any point in this method a point does not have any other points within w of it, we simply partition it as its own partition.

Beyond this, the algorithm simply proceeds as the others to construct a hierarchy. Our ball radius starts as $w = \Delta/2$ for the top level and is scaled by $1/2$ at each recursive step. We recurse on each partition until all partitions are empty or singletons, and we create an edge of weight $\sqrt{r}w$ from each partition to its parent partition. With this hierarchy as our final tree embedding, it is not hard to see that the distance between any two points can be calculated by the number of levels of the hierarchy in which the two points are separated. If at any level they are separated, we add $\sqrt{r}w$ to their distance.

It is not too difficult to see how this method generalizes both the random shifted grid and ball partitions. Definitionally, if $r = 1$, then there is a single bucket containing all dimensions, and thus we are simply working in the original space. Then for any iteration of ball partitioning, two points are grouped together if they are captured by the same ball. Therefore, for $r = 1$, we are simply running the

ball partitioning algorithm. If $r = d$ and if we let the ball radius be half the cell width, then each dimension is given its own bucket. When we run a ball partition on each bucket, or each 1-dimensional space, independently, we are just shifting that coordinate and partitioning that dimension into equally-sized intervals. Intersecting the partitions formed by partitioning dimensions like this is the same as defining d -dimensional cells in a grid based off a random shift vector composed of the random shifts of each individual dimension. Thus, we get precisely the random shifted grid.

Now that we have discussed the algorithm, we move on to the desired result. Note that all proofs can be found in the appendix. We ultimately show that, as long as we cover all points in each level (this is discussed probabilistically, by setting a high value of U , in Section 4):

Theorem 2. Consider a set of n points $P \subseteq [\Delta]^d$ for $\Delta \in \mathbb{Z}_{\geq 1}$ and a parameter $r \in [d]$. Algorithm 1 computes a weighted spanning tree T over P such that $\forall p, q \in P$, $\|p - q\|_2 \leq \text{dist}_T(p, q)$ and $E_T[\text{dist}_T(p, q)] \leq O(\sqrt{d} \cdot r \cdot \log \Delta) \cdot \|p - q\|_2$.

The first property, the lower bound on the tree distance, is the notion of dominance. The second defines the amount of distortion resulting from the metric. Both are commonly required in popular probabilistic tree embeddings. It turns out that the first is much simpler to show for our algorithm:

Lemma 2. For any two points $p, q \in P$, $\|p - q\|_2 \leq \text{dist}_T(p, q)$.

The next goal is to prove the distortion. Crucially, to show this, we must find a relationship between the distance between two points and whether or not they are in the same partition at some level. This brings up the following lemma, which reveals an interesting property: the probability that two points are separated at a level can be bounded in a way that is independent of r , however the diameter of a partition relative to P is bounded by $\sqrt{r} \cdot w$.

Lemma 1. Consider a hybrid partitioning with parameters $w \in \mathbb{R}_{>0}$ and $r \leq d$ in the d -dimensional Euclidean space. For any two points $p, q \in \mathbb{R}^d$, the probability that p and q are assigned to different partitions is at most $O\left(\sqrt{d} \cdot \frac{\|p - q\|_2}{w}\right)$. If $p, q \in \mathbb{R}^d$ are assigned to the same partition, $\|p - q\|_2 \leq O(\sqrt{r} \cdot w)$.

Since this lemma is quite extensive, we break it down into two parts. The more interesting part is considering the probability that two points are separated in the partitioning at a certain level. We simply consider a ball partitioning:

Lemma 3. Consider a ball partitioning over a set of points $P \subset \mathbb{R}^k$. For any two points $p, q \in P$, the probability that p and q are assigned to different partitions in a level with scale w is at most $O\left(\sqrt{k} \frac{\|p - q\|_2}{w}\right)$.

In order to solve this, we consider a random variable $I_{q,p}$ that indicates if q and p exist in the same ball in a partitioning. This can be used to determine the probability of separation, but we can also relate it to the volume of the intersection of balls centered at these points with radius w . Eventually, we see that the probability of separation boils down to the probability that a random unit vector falls into the bound of a cap surface intersected with the surface of a sphere. In order to bound this, we need a slight detour into more pure geometric arguments, Lemma 4.

In particular, the following lemma shows that the probability that a random vector is near the equator of a unit sphere has a good upper bound.

Lemma 4. Let $u \in \mathbb{R}^d$ be a random vector drawn uniformly from a unit sphere. Then for any $D, w \in \mathbb{R}_{\geq 0}$, we have

$$\Pr[|u_1| \leq D/(2w)] = O\left(\sqrt{d} \cdot \frac{D}{w}\right).$$

While this lemma is close to the result we need, it unfortunately deals with vectors drawn from a voluminous shape, whereas we are interested in the probability that a random vector drawn from a surface falls within the bounds of intersected surfaces. A slight adaptation yields the following:

Lemma 5. Let $v \in \mathbb{R}^d$ be a random vector drawn uniformly from a unit ball. Then for any $D, w \in \mathbb{R}_{\geq 0}$, we have

$$\Pr[|v_1| \leq D/(2w)] = O\left(\sqrt{d} \cdot \frac{D}{w}\right).$$

For two points to not be separated within some bucket in the partitioning, they must either be grouped together or left uncovered. This allows us to relate the probability of separation to the intersection of geometric surfaces. Ultimately, we can then derive a bound from Lemma 3 using Lemma 5 (see the proof of Lemma 3 in the appendix. Since two points separated on a level must be separated in some bucket, the separation probability is bounded by a union bound over the probability of separation in each bucket, yielding Lemma 1. As we have already showed the domination result, all we need is to show that the expected distortion is small, which is directly related to the probability that two points are separated on any level. This concludes Theorem 2.

4 TREE EMBEDDING IN MPC

In this section we show how to implement Algorithm 1 in $O(1)$ of rounds in the MPC model using the total space $O(n \cdot d + n \cdot \log \Delta \cdot \log n \cdot \log \log n)$. Specifically, we prove Theorem 1.

Theorem 1. Consider a set of n points $P \subseteq [\Delta]^d$ for $\Delta \in \mathbb{Z}_{\geq 1}$. There is an $O(1)$ -round randomized MPC algorithm which computes a weighted spanning tree T over P when it succeeds, such that $\forall p, q \in P$,

- (1) $\text{dist}_T(p, q) \geq \|p - q\|_2$,
- (2) $E_T[\text{dist}_T(p, q)] \leq O(\sqrt{\log n} \cdot \log \Delta \cdot \sqrt{\log \log n}) \cdot \|p - q\|_2$.

The success probability is at least $1 - 1/\text{poly}(n)$. The algorithm uses $O(n \cdot d + n \log n \cdot (\log \Delta \cdot \log \log n + \min(d, \log^2 n)))$ total space and each machine holds $O((nd)^\epsilon)$ local space for an arbitrary constant $\epsilon \in (0, 1)$. If the algorithm fails, it reports failure.

At a high level, our algorithm consists of four main steps:

- (1) Using our fast Johnson-Lindenstrauss (see Section 5), embed the data into $O(\log n)$ dimensions. This is the first $O(1)$ rounds of the algorithm.
- (2) We group dimensions into r buckets and generate grids for each bucket and distribute the grids and points among the machines. This requires 1 round on 1 machine.
- (3) We compute the hierarchical tree using ball partitioning. This requires 1 round of parallel computation.

- (4) For each node, we compute the path-to-root in the hybrid partitioning (to construct the final tree). This also requires 1 round of parallel computation.

In order to create the hierarchy in parallel, the grid of balls will be shared among all machines. Andoni [4] lets us bound the number of grids needed to cover the entire space:

Lemma 6 ([4] Section 3.2.2). *Consider a d -dimensional space $\mathbb{R}^d$, and fix some $\delta > 0$. Let G be a regular infinite grid of balls of radius w placed at coordinates $4w \cdot \mathbb{Z}^d$. Define G_u , for $u \in \mathbb{N}$, as $G_u = G + s_u$, where $s_u \in [0, 4w]^d$ is a uniformly selected random shift of the grid G . If $U_d = 2^{O(d \log d)} \log 1/\delta$, then the grids $G_1, G_2, \dots, G_{U_d}$ cover the entire space $\mathbb{R}^d$, with probability at least $1 - \delta$.*

Since we will be using grids to cover the whole space for each bucket and level we need the following lemma:

Lemma 7. *Consider n points over a d -dimensional space, and fix some parameter $\epsilon > 0$, $\delta > 0$. Define U as the number of grids of balls used in the hybrid partitioning. By setting $U = 2^{O((d/r) \log(d/r))} \cdot \log(\frac{r \log \Delta}{\delta})$, hybrid partitioning covers the whole space with probability at least $1 - \delta$.*

For standard ball partitioning the number of grids needed to cover the whole space would be too large. More specifically, for standard ball partitioning of a d -dimensional space we need $2^{O(d \log d)}$ grids. Even after reducing the data into $O(\log n)$ dimensions, this would result in $2^{O(\log n \log \log n)} = n^{O(\log \log n)}$ dimensions which would be too large considering we have $O(n^\epsilon)$ space per machine. Hence the need to use *hybrid partitioning*.

To do so first we transform the data points from d -dimensional space to $O(\log n)$ dimensional space using our fast Johnson-Lindenstrauss transform. We fix $r = 2/\epsilon \cdot \log \log n$. Then we group the dimensions into r buckets and distribute them among the machines, each machine holding $\frac{n^\epsilon}{\log \Delta}$ points. This is because the ball partitioning needs $\log \Delta$ space per point to store the hierarchy, assuming we store the path from a vertex to the root.

For each bucket j we generate sets of grids $\{G_1^j, G_2^j, \dots, G_{\log \Delta}^j\}$ and distribute them to the corresponding machines that hold points from this bucket. Where $G_i^j = \{B_1, B_2, \dots, B_U\}$ is the set of grids that cover the space in bucket j in level 2^i . Each B_k is a partitioning from a single randomly shifted instance of a ball partition.

Define U as the number of grids used to cover the space in our hybrid partitioning.

Lemma 8. *Consider n points over a d -dimensional space where $d = O(\log n)$, and a hybrid partitioning with $r = 2/\epsilon \cdot \log \log n$ buckets. The space used per machine to store the set of grids is $O(n^\epsilon)$ as long as $\delta = \Omega(1/\text{poly}(n))$.*

This proves all grids can be stored on each machine, and thus the local computation can proceed in constant rounds to complete Theorem 1.

5 MPC FAST JOHNSON-LINDENSTRAUSS

The Johnson Lindenstrauss transform, which maps any high-dimensional data into logarithmic dimensions with arbitrarily small distortion constant, is a foundational method for (potentially) significant reductions in data dimension in the sequential setting. Unfortunately,

Algorithm 2 MPC Hybrid Partitioning

Input: Point set $P \subseteq [\Delta]^d$ and a parameter $r \in [d]$ (after dimension reduction, e.g., see Section 5).

Output: T , a tree embedding of P

// Run on a single machine:

Split the dimensions $[d]$ into $r = O(\log \log n)$ buckets // As in the first step of Algorithm 1

Let $U = 2^{O((d/r) \log(d/r))} \cdot \log(r \log \Delta)$ // This comes from Lemma 7

For each $j \in [r]$: $G^{(j)} = \text{BuildGrids}(P^{(j)}, r, u)$

Send G to all other machines

Partition P into parts P_i with $|P_i| = O(n^\epsilon / \log \Delta)$ and send P_i to machine m_i

// In parallel:

for each machine m_i **do**

For each $j \in [r]$: $T_j \leftarrow \text{BallPart}(P_i^{(j)}, G^{(j)})$

If any ball partitionings failed, halt and report failure

for each $p \in P_i$ **do**

for $j \in [r]$ **do**

Find $\text{path}^j(p) \leftarrow (v_0^{(j)}, v_1^{(j)}, \dots, v_{\log \Delta}^{(j)}, p^{(j)})$, the path from p to the root of T_j

For $i \in [\log \Delta]$: $\text{path}_i(p) \leftarrow (v_i^{(1)}, v_i^{(2)}, \dots, v_i^{(r)})$, the tuple for the i th element on p 's path for all T_j

$\text{path}(p) \leftarrow (\text{path}_1(p), \text{path}_2(p), \dots, \text{path}_{\log \Delta}(p), p)$, the path from p to the root of T

Let T_i be the union of $\text{path}(p)$ for all $p \in P_i$

return T_i as a part of the output tree T (implicitly, T is the union of all returned T_i s)

in the massively parallel setting, this effectively becomes a general matrix multiplication problem, which does not meet the round and space complexity goals of many MPC algorithms. Particularly, we can achieve this either using $O(ndk) = O(nd \log n)$ total space in a constant number of rounds [31] or using linear space $O(nd)$ in $O(\log n)$ rounds.

To prove our main result, Theorem 1, we require first applying the Johnson Lindenstrauss transform to reduce the dimensionality of our data, then we apply our methods from Section 4. The transform must be applied first because otherwise the number of balls required to partition the entire space with sufficiently high probability is too large, and they cannot be fit within our total space requirements.

Recall that our final results manage to shave off the additional logarithmic factor in the total space. Namely, running an $O(nd \log n)$ total space implementation of the Johnson Lindenstrauss transform would increase our total space. Therefore, we propose a solution to the dimension-reducing problem in the MPC model based on a more recent iteration of the Johnson Lindenstrauss transform by Ailon and Chazelle [2] that uses linear total space.

Consider a dataset of n points in a d -dimensional Euclidean space. The Fast Johnson Lindenstrauss Transform (FJLT), developed by Ailon and Chazelle [2], is a quickly computed transformation that maps the input points into a k -dimensional space while preserving distances within a $(1 \pm \xi)$ factor. Here, $k = c\xi^{-2} \log n$ for some constant c .

They show that the FJLT, $\phi(x) = k^{-1}PHDx$, for some input x can be computed relatively quickly. Matrices P , H , and D are designed as follows:

- (1) P is a $k \times d$ matrix with $P_{ij} = 0$ with probability $1 - q$, otherwise it is sampled from the Gaussian distribution $N(0, q^{-1})$. We let q take value:

$$q = \Theta\left(\min\left(\frac{\log^2 n}{d}, 1\right)\right)$$

- (2) H is a normalized $d \times d$ Walsh-Hadamard matrix. In other words, it is the d -dimensional discrete Fourier transform. In particular, $H_{i,j} = d^{-1/2} \cdot (-1)^{\langle i-1, j-1 \rangle}$ where $\langle i-1, j-1 \rangle$ is the bitwise inner product of the binary representations of $i-1$ and $j-1$.
- (3) D is a random diagonal $d \times d$ matrix where $D_{i,i} = -1$ with probability 0.5 and $D_{i,i} = 1$ otherwise.

We now propose a method to parallelize this algorithm in the MPC model as shown in Algorithm 3. We consider the case where each machine has local space $O((nd)^\epsilon)$ for some arbitrary constant $\epsilon \in (0, 1)$. The algorithm takes an input A which is a $d \times n$ matrix. To compute $PHD(A)$, we simply apply one matrix at a time. To compute $D(A)$, since D is diagonal, we can use $O(1)$ rounds and $O(d)$ total space to generate all $D_{i,i}$. To multiply D with A , since D is diagonal, each of the resulting nd entries in DA form a simple binary multiplication problem. We simply allocate $(nd)^{1-\epsilon}$ machines to do $O((nd)^\epsilon)$ computations each. To apply H , we can simply apply the fast Fourier transform in the MPC model introduced by Hajiaghayi, Saleh, Seddighin, and Sun [45]. This step takes $O(1/\epsilon)$ rounds. Finally, we generate P in a similar way to D , by allocating $O((nd)^\epsilon)$ entries per machine and randomly generating numbers. To apply P , we can bound the total number of multiplications by the results of Ailon and Chazelle, and then distribute additions across machines iteratively to compute the matrix in $O(1/\epsilon)$ rounds.

Theorem 3. Consider a set of points $P = \{p_1, p_2, \dots, p_n\} \subset \mathbb{R}^d$. Let $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^k$ be Fast Johnson Lindenstrauss Transform with $k = \Theta(\xi^{-2} \log n)$ for $\xi \in (0, 0.5)$. There is an MPC algorithm which outputs $\phi(p_1), \phi(p_2), \dots, \phi(p_n)$ in $O(1)$ rounds. In addition, the total space of the algorithm is at most $O(nd + \xi^{-2} n \log^3 n)$ and each machine holds $O((nd)^\epsilon)$ local space for an arbitrary constant $\epsilon \in (0, 1)$.

PROOF. Let $A \in \mathbb{R}^{d \times n}$ be the concatenation of points $p_1, p_2, \dots, p_n$. Then, the goal is to compute $P \cdot H \cdot D \cdot A$. First, we show how to compute DA (a $d \times n$ -dimensional matrix). Clearly we can use $O(1)$ rounds and $O(d)$ total space to generate the entries on the diagonal of D . Note that we can easily compute each element $(DA)_{i,j} = D_{i,i}A_{i,j}$. This is a total of nd computations. So we can do it on $(nd)^{1-\epsilon}$ machines with $(nd)^\epsilon$ local space per machine in $O(1)$ rounds.

Next, we compute $H(DA)$ (a $d \times n$ -dimensional matrix). Note that H is just the d -dimensional DFT. We can then utilize the d -dimensional MPC FFT algorithm [45] and apply it to each column of DA . For any $\epsilon \in (0, 1)$, this requires $O(d^\epsilon)$ machines, $O(d^{1-\epsilon})$ memory, and $O(1/\epsilon)$ rounds. Thus, we can compute all columns of DA in the MPC model using $O((nd)^\epsilon)$ space per machine and $O((nd)^{1-\epsilon})$ machines in $O(1/\epsilon)$ rounds.

Finally, we compute $P(HDA)$ (a $k \times n$ -dimensional matrix). Just as in Ailon and Chazelle, we see that the number of nonzero values, or $|P|$, is $\sim \text{Binom}(dk, q)$, which means:

$$E[|P|] = nkq = O\left(d\xi^{-2} \log^3(n)/d\right) = O(\log^3(n)/\xi^2)$$

And then by the Markov inequality, we have $|P| = O(\log^3(n)/\xi^2)$ with probability at least 0.99. This means we only have $O(\xi^{-2} \log^3 n)$ values in P with probability at least 0.99. For each value in P , we multiply it by one row of HDA of length n for a total of $O(n\xi^{-2} \log^3 n)$ computations. Beyond this, some of these values need to be added to find values in $PHDA$, but that will not exceed $O(n\xi^{-2} \log^3 n)$. Thus, these computations can be done in $O(1)$ rounds as long as the total space is at least $\Omega(n\xi^{-2} \log^3 n)$. Finally, there are at most d additions required to define a single entry, then we can divide this into $d^{1-\epsilon}$ sets of additions of size d^ϵ . We can then pack all these sets of computations of size at most d^ϵ into machines with memory $O(d^\epsilon)$. This requires at most $O(d^{1-\epsilon})$ machines. Then we perform the computations. To find the values for entries that require n computations, this will require $O(1/\epsilon)$ rounds. That completes the computation. $\square$

Algorithm 3 FJLT in MPC

Input: A , a matrix of n d -dimensional vectors, and a parameter $\xi \in (0, 0.5)$

Output: $\phi(A)$

for $i \in [d]$ **in parallel do**

Let $D_{i,i}$ be 1 or -1 with probabilities 1/2

for $i \in [d], j \in [n]$ **in parallel do**

$(DA)_{i,j} \leftarrow D_{i,i}A_{i,j}$

Let $H(DA) \leftarrow \text{FFT}(DA, \epsilon)$

//MPC algorithm for fast Fourier transform [45]. Each machine holds $O((nd)^\epsilon)$ space.

Let M be the set of multiplications of nonzero entries in P and entries in HDA

for Assign M to $(nd)^{1-\epsilon} \xi^{-2} \log^3 n/d$ machines **in parallel do**

Compute multiplication $m \in M$ locally.

Let $\mathcal{A}$ be the set of additions of m outputs in $P(HDA)$

while $\mathcal{A}$ is nonempty **do**

for Pack large contiguous chunks of $a \in \mathcal{A}$ into $(nd)^\epsilon$ per machine **in parallel do**

Compute a and simplify $\mathcal{A}$

Store results of $\mathcal{A}$ in $\phi(A) = P(HDA)$

REFERENCES

- [1] Kook Jin Ahn and Sudipto Guha. 2015. Access to Data and Number of Iterations: Dual Primal Algorithms for Maximum Matching under Resource Constraints. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms*. 202–211.
- [2] Nir Ailon and Bernard Chazelle. 2006. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*. 557–563.
- [3] Noga Alon, Richard M. Karp, David Peleg, and Douglas B. West. 1995. A Graph-Theoretic Game and Its Application to the k-Server Problem. *SIAM J. Comput.* (1995), 78–100.
- [4] Alexandr Andoni. 2009. *Nearest Neighbor Search: the Old, the New, and the Impossible*. Ph. D. Dissertation. Massachusetts Institute of Technology, Cambridge, MA, USA.

- [5] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. 2014. Parallel algorithms for geometric graph problems. In *Symposium on Theory of Computing*. ACM, 574–583.
- [6] Alexandr Andoni, Zhao Song, Clifford Stein, Zhengyu Wang, and Peilin Zhong. 2018. Parallel Graph Connectivity in Log Diameter Rounds. In *59th IEEE Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 674–685.
- [7] Alexandr Andoni, Clifford Stein, and Peilin Zhong. 2019. Log Diameter Rounds Algorithms for 2-Vertex and 2-Edge Connectivity. In *46th International Colloquium on Automata, Languages, and Programming*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 14:1–14:16.
- [8] Alexandr Andoni, Clifford Stein, and Peilin Zhong. 2020. Parallel approximate undirected shortest paths via low hop emulators. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. 322–335.
- [9] Sanjeev Arora. 1997. Nearly Linear Time Approximation Schemes for Euclidean TSP and other Geometric Problems. In *38th Annual Symposium on Foundations of Computer Science*. 554–563.
- [10] Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. 2019. Coresets Meet EDCS: Algorithms for Matching and Vertex Cover on Massive Graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete*. 1616–1635.
- [11] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. 2019. Sublinear Algorithms for $\Delta+1$ Vertex Coloring. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete*. 767–786.
- [12] Sepehr Assadi, Xiaorui Sun, and Omri Weinstein. 2019. Massively Parallel Algorithms for Finding Well-Connected Components in Sparse Graphs. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed*. 461–470.
- [13] Bahman Bahmani, Ashish Goel, and Kamesh Munagala. 2014. Efficient Primal-Dual Graph Algorithms for MapReduce. In *Algorithms and Models for the Web Graph - 11th International Workshop, WAW 2014, Beijing, China, December 17-18, 2014, Proceedings (Lecture Notes in Computer Science, Vol. 8882)*, Anthony Bonato, Fan Chung Graham, and Pawel Pralat (Eds.). Springer, 59–78.
- [14] Yair Bartal. 1996. Probabilistic Approximations of Metric Spaces and Its Algorithmic Applications. In *37th Annual Symposium on Foundations of Computer Science*. 184–193.
- [15] Yair Bartal. 1998. On Approximating Arbitrary Metrics by Tree Metrics. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory*. 161–168.
- [16] MohammadHossein Bateni, Soheil Behnezhad, Mahsa Derakhshan, Mohammad-Taghi Hajiaghayi, Raimondas Kiveris, Silvio Lattanzi, and Vahab S. Mirrokni. 2017. Affinity Clustering: Hierarchical Clustering at Scale. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*. 6864–6874.
- [17] MohammadHossein Bateni, Soheil Behnezhad, Mahsa Derakhshan, Mohammad-Taghi Hajiaghayi, and Vahab S. Mirrokni. 2018. Brief Announcement: MapReduce Algorithms for Massive Trees. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic (LIPIcs, Vol. 107)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 162:1–162:4. <https://doi.org/10.4230/LIPIcs.ICALP.2018.162>
- [18] Paul Beame, Paraschos Koutiris, and Dan Suciu. 2017. Communication steps for parallel query processing. *Journal of the ACM (JACM)* 64, 6 (2017), 1–58.
- [19] Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Marina Knittel, and Hamed Saleh. 2019. Streaming and Massively Parallel Algorithms for Edge Coloring. In *27th Annual European Symposium on Algorithms*. 15:1–15:14.
- [20] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, Vahab Mirrokni, and Warren Schudy. 2019. Massively parallel computation via remote memory access. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*. 59–68.
- [21] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, and Vahab S. Mirrokni. 2019. Near-Optimal Massively Parallel Graph Connectivity. In *60th IEEE Annual Symposium on Foundations of Computer Science*. 1615–1636.
- [22] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, Vahab S. Mirrokni, and Warren Schudy. 2020. Parallel Graph Algorithms in Constant Adaptive Rounds: Theory meets Practice. *Proc. VLDB Endow.* (2020), 3588–3602.
- [23] Soheil Behnezhad, MohammadTaghi Hajiaghayi, and David G. Harris. 2019. Exponentially Faster Massively Parallel Maximal Matching. In *60th IEEE Annual Symposium on Foundations of Computer Science*. 1637–1649.
- [24] Guy E. Blelloch, Anupam Gupta, and Kanat Tangwongsan. 2012. Parallel probabilistic tree embeddings, k-median, and buy-at-bulk network design. In *24th ACM Symposium on Parallelism in Algorithms and Architectures*. 205–213.
- [25] Craig Chambers, Ashish Raniwala, Frances Perry, Stephen Adams, Robert R. Henry, Robert Bradshaw, and Nathan Weizenbaum. 2010. FlumeJava: easy, efficient data-parallel pipelines. In *Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation*, Benjamin G. Zorn and Alexander Aiken (Eds.). 363–375.
- [26] Hubert Tsz-Hong Chan, Anupam Gupta, Bruce M. Maggs, and Shuheng Zhou. 2005. On hierarchical routing in doubling metrics. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*. SIAM, 762–771. <http://dl.acm.org/citation.cfm?id=1070432.1070540>
- [27] Moses Charikar, Chandra Chekuri, Ashish Goel, Sudipto Guha, and Serge A. Plotkin. 1998. Approximating a Finite Metric by a Small Number of Tree Metrics. In *39th Annual Symposium on Foundations of Computer Science*. 379–388.
- [28] Xi Chen, Rajesh Jayaram, Amit Levi, and Erik Waingarten. 2020. An Improved Analysis of the Quadtree for High Dimensional EMD. (2020).
- [29] Artur Czumaj, Jakub Lacki, Aleksander Madry, Slobodan Mitrovic, Krzysztof Onak, and Piotr Sankowski. 2019. Round compression for parallel matching algorithms. *SIAM J. Comput.* 49, 5 (2019), STOC18–1.
- [30] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.
- [31] Alessandro Epasto, Mohammad Mahdian, Vahab S. Mirrokni, and Peilin Zhong. 2021. Massively Parallel and Dynamic Algorithms for Minimum Size Clustering. CoRR abs/2106.02685 (2021). arXiv:2106.02685 <https://arxiv.org/abs/2106.02685>
- [32] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. 2003. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*. 448–455.
- [33] Apache Software Foundation. [n. d.]. *Hadoop*. <https://hadoop.apache.org/>.
- [34] Stephan Friedrichs and Christoph Lenzen. 2018. Parallel metric tree embedding based on an algebraic view on moore-bellman-ford. *Journal of the ACM (JACM)* 65, 6 (2018), 1–55.
- [35] Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrovic, and Ronitt Rubinfeld. 2018. Improved Massively Parallel Computation Algorithms for MIS, Matching, and Vertex Cover. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed*. 129–138.
- [36] Mohsen Ghaffari, Silvio Lattanzi, and Slobodan Mitrovic. 2019. Improved Parallel Algorithms for Density-Based Network Clustering. In *Proceedings of the 36th International Conference on Machine Learning, ICLR 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 2201–2210.
- [37] Mohsen Ghaffari and Jara Uitto. 2019. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1636–1653.
- [38] Michael T Goodrich, Nodari Sitchinava, and Qin Zhang. 2011. Sorting, searching, and simulation in the mapreduce framework. In *International Symposium on Algorithms and Computation*. Springer, 374–383.
- [39] Anupam Gupta, Mohammad Taghi Hajiaghayi, and Harald Räcke. 2006. Oblivious network design. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*. ACM Press, 970–979. <http://dl.acm.org/citation.cfm?id=1109557.1109665>
- [40] Anupam Gupta, Robert Krauthgamer, and James R. Lee. 2003. Bounded Geometries, Fractals, and Low-Distortion Embeddings. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*. IEEE Computer Society, 534–543.
- [41] Anupam Gupta, Ilan Newman, Yuri Rabinovich, and Alistair Sinclair. 2004. Cuts, Trees and l_1 -Embeddings of Graphs. *Comb.* 24, 2 (2004), 233–269. <https://doi.org/10.1007/s00493-004-0015-x>
- [42] MohammadTaghi Hajiaghayi and Marina Knittel. 2020. Matching Affinity Clustering: Improved Hierarchical Clustering at Scale with Guarantees. In *Proceedings of the 19th International Conference on Autonomous Agents*. 1864–1866.
- [43] MohammadTaghi Hajiaghayi, Marina Knittel, Jan Olkowski, and Hamed Saleh. 2022. Adaptive Massively Parallel Algorithms for Cut Problems. In *SPAA '22: 34th ACM Symposium on Parallelism in Algorithms and Architectures*, Kunal Agrawal and I-Ting Angelina Lee (Eds.). ACM, 23–33.
- [44] MohammadTaghi Hajiaghayi, Marina Knittel, Hamed Saleh, and Hsin-Hao Su. 2022. Adaptive Massively Parallel Constant-Round Tree Contraction. , 83:1–83:23 pages.
- [45] MohammadTaghi Hajiaghayi, Hamed Saleh, Saeed Seddighin, and Xiaorui Sun. 2021. Massively Parallel Algorithms for String Matching with Wildcards. (2021).
- [46] William Johnson and Joram Lindenstrauss. 1984. Extensions of Lipschitz maps into a Hilbert space. *Contemp. Math.* (1984), 189–206.
- [47] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. 2010. A model of computation for MapReduce. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. SIAM, 938–948.
- [48] R. Karp. 1989. A 2k-competitive algorithm for the circle.
- [49] Goran Konjevod, R. Ravi, and F. Sibel Salman. 2001. On approximating planar metrics by tree metrics. *Inf. Process. Lett.* (2001), 213–219.
- [50] James R Lee and Assaf Naor. 2004. Embedding the diamond graph in L_p and dimension reduction in L_1 . *Geometric & Functional Analysis GAFA* 14, 4 (2004), 745–747.
- [51] Assaf Naor and Gideon Schechtman. 2006. Planar Earthmover is not in L_1 . In *47th Annual IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society, 655–666.
- [52] Yuri Rabinovich and Ran Raz. 1998. Lower Bounds on the Distortion of Embedding Finite Metric Spaces in Graphs. *Discret. Comput. Geom.* 19, 1 (1998), 79–94. <https://doi.org/10.1007/PL00009336>
- [53] Satish Rao. 1999. Small Distortion and Volume Preserving Embeddings for Planar and Euclidean Metrics. In *Proceedings of the Fifteenth Annual Symposium on*

- Computational Geometry* (Miami Beach, Florida, USA) (SCG '99). Association for Computing Machinery, New York, NY, USA, 300–306. <https://doi.org/10.1145/304893.304983>
- [54] Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. 2016. Shuffles and Circuits: (On Lower Bounds for Modern Parallel Computation). In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms*. 1–12.
- [55] Tom White. 2012. *Hadoop: The definitive guide*. " O'Reilly Media, Inc."
- [56] Grigory Yaroslavltssev and Adithya Vadapalli. 2018. Massively Parallel Algorithms and Hardness for Single-Linkage Clustering under ℓ_p Distances. In *Proceedings of the 35th International Conference on Machine Learning*. 5596–5605.
- [57] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. 2016. Apache Spark: a unified engine for big data processing. *Commun. ACM* (2016), 56–65.