Parallel Algorithms for Geometric Graph Problems[*]

Alexandr Andoni
Microsoft Research

Aleksandar Nikolov
Rutgers University

Krzysztof Onak[†]
IBM Research

Grigory Yaroslavtsev [‡]
ICERM, Brown University

ABSTRACT

We give algorithms for geometric graph problems in the modern parallel models such as MapReduce. For example, for the Minimum Spanning Tree (MST) problem over a set of points in the two-dimensional space, our algorithm computes a $(1+\epsilon)$-approximate MST. Our algorithms work in a *constant* number of rounds of communication, while using total space and communication proportional to the size of the data (linear space and near linear time algorithms). In contrast, for general graphs, achieving the same result for MST (or even connectivity) remains a challenging open problem [9], despite drawing significant attention in recent years.

We develop a general algorithmic framework that, besides MST, also applies to Earth-Mover Distance (EMD) and the transportation cost problem. Our algorithmic framework has implications beyond the MapReduce model. For example it yields a new algorithm for computing EMD cost in the plane in near-linear time, $n^{1+o_\epsilon(1)}$. We note that while recently [33] have developed a near-linear time algorithm for $(1+\epsilon)$-approximating EMD, our algorithm is fundamentally different, and, for example, also solves the transportation (cost) problem, raised as an open question in [33]. Furthermore, our algorithm immediately gives a $(1+\epsilon)$-approximation algorithm with n^δ space in the streaming-with-sorting model with $1/\delta^{O(1)}$ passes. As such, it is tempting to conjecture that the parallel models may also constitute a concrete playground in the quest for efficient algorithms for EMD (and other similar problems) in the vanilla *streaming model*, a well-known open problem.

[*]A full version of this paper is available at `http://arxiv.org/abs/1401.0042`

[†]Supported by the Simons Postdoctoral Fellowship. Research initiated while at CMU.

[‡]Research initiated at Microsoft Research. The author was also supported by the Institute Postdoctoral Fellowship in Mathematics at Brown University, Institute for Computational and Experimental Research in Mathematics.

STOC '14, May 31 - June 03 2014, New York, NY, USA
Copyright 2014 ACM 978-1-4503-2710-7/14/05...$15.00.
http://dx.doi.org/10.1145/2591796.2591805

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; G.2.2 [**Discrete Mathematics**]: Graph Theory

Keywords

parallel computation, minimum spanning tree, earth-mover distance, MapReduce

1. INTRODUCTION

Over the past decade a number of parallel systems have become widely successful in practice. Examples of such systems include MapReduce [13, 14], Hadoop [39], and Dryad [26]. Given these developments, it is natural to revisit algorithmics for parallel systems and ask what new algorithmic or complexity ideas Theoretical Computer Science can contribute to this line of research (and engineering) efforts.

Two theoretical questions emerge: 1) What models capture well the capabilities of the existing systems? 2) What new algorithmic ideas can we develop for these models? Addressing Question 1, researchers [17, 27, 20, 9] have proposed a model which balances simplicity and relevance to practice. We describe this model in Section 1.1. As for Question 2, while there already exist a few algorithms adapted or designed for this model (see Section 1.3), we feel that many more powerful algorithmic ideas are still waiting to be developed.

It is natural to ask first: do we really need new algorithmics here? A lot of fundamental research on parallel algorithms was conducted in the 1980s and 1990s, most notably in the PRAM model. One may hope to reuse that line of research in the new models. Indeed, the works of [27, 20] have shown that one can simulate PRAM algorithms in MapReduce with minimal slow-down. Is there anything new about the modern parallel systems?

The answer is that the parameters of the new models are such that we can hope for *faster* algorithms than those possible in the PRAM model. The models allow for interleaving parallel and sequential computation: in a single step, a machine can perform arbitrary polynomial time computation on its local input. The time cost of the algorithm is then measured in the number of *rounds of communication* between machines. This makes it possible to achieve *constant* parallel time for interesting problems, while in the PRAM model functions that depend on the entire input generally require logarithmic or larger parallel time. For example, even computing the XOR of n variables requires near-logarithmic parallel-time on the most powerful CRCW PRAMs [8]. In contrast, in the new models, which are similar to a n^α-fan-in circuit, one can trivially solve XOR in $O(1/\alpha)$ parallel time. Indeed, the MapReduce models rather fall under the blanket of the generic Bulk Synchronous Parallel (BSP) model [37], though this model has a number of parameters, and as such has not been thoroughly explored. In particular, few solutions to even very fundamental problems are known in the BSP models (see, e.g., [19] for a sorting algorithm). The new models instead focus on a specific range of parameters and tradeoffs, making analysis more tractable.

The previous work on MapReduce models identifies a captivating and challenging problem: connectivity in a sparse graph. While this problem has a classic logarithmic time PRAM algorithm [34], we do not know whether we can solve it faster in the new models [27]. For this particular problem, though, recent results show logarithmic lower bounds for restricted algorithms [9], suggesting that the negative answer may be more plausible.

Synopsis of contributions.

In this work, we focus on basic graph problems in the *geometric* setting, and show we can achieve $1+\epsilon$ approximation in a constant number of rounds. In fact, we develop a common algorithmic framework applicable to graph questions such as Minimum Spanning Tree and Earth-Mover Distance. Thus, while it may be hard to speed up standard graph algorithms (without geometric context) in MapReduce-like models [9], our results suggest that speedups can be obtained if we manage to represent the graph in a geometric fashion (e.g., in a similarity space).

Our framework turns out to be quite versatile and has implications beyond parallel computing. For example it yields a new algorithm for computing EMD (cost) in the plane in near-linear time, $n^{1+o_\epsilon(1)}$. We note that while recently [33] have developed a near-linear time algorithm for $(1+\epsilon)$-approximating EMD, our algorithm is different, and, for example, also solves the transportation (cost) problem, raised as an open question in [33]. In particular, our algorithm uses little of the combinatorial structure of EMD, and essentially relies only on an off-the-shelf LP solver. In contrast, [33] intrinsically exploit the combinatorial structure, together with carefully designed data structures to obtain a $O_\epsilon(n \log^{O(1)} n)$ time algorithm. Their approach, however, seems hard to parallelize. Another consequence is a $(1+\epsilon)$-approximation algorithm in the *streaming-with-sorting* model, with n^δ space and $1/\delta^{O(1)}$ passes.

1.1 The Model

We adopt the most restrictive MapReduce-like model among [27, 20, 9]. Following [9], we call the model *Massively Parallel Communication* or MPC (although we explicitly consider the local sequential running times as well).

Suppose we have m machines (processors) each with space s, where n is the size of the input and $m \cdot s = O(n)$. Thus, the total space in the system is only a constant factor more than the input size, allowing for minimal replication.

The computation proceeds in rounds. In each round, a machine performs local computation on its data (of size s), and then sends messages to other machines for the next round. Crucially, the total amount of communication sent or received by a machine is bounded by s, its space. For example, a machine can send one message of size s, or s messages of size 1. It cannot, however, broadcast a size-s message to every machine. In the next round, each machine treats the received messages as the input for the round.

The main complexity measure is the number of rounds R required to solve a problem, which we consider to be the "parallel time" of the algorithm. Some related models, such as BSP, also consider the sequential running time of a machine in a round. We de-emphasize this aspect, as we consider the information-theoretic question of understanding the round complexity to be the first-order business here. In particular, the restriction on space alone (i.e., with *unbounded* computation per machine) already appears to make certain problems require a super-constant number of rounds, including the connectivity in sparse graphs. Nevertheless, it is natural to minimize the local running time, and indeed our (local) algorithms run in time polynomial in s, leading to $O(ns^{O(1)})$ overall work.

What are good values of s and R? As in [27, 20], we assume that space s is polynomial in n, i.e., $s = n^\alpha$ for some $\alpha > 0$. We consider this a justified choice since even under the natural assumption that $s \geq m$ (i.e., each machine has an index of all other machines), we immediately obtain that $s \geq \sqrt{n}$.[1]

Our goal is to obtain $R = \text{poly}(\log_s n) = O(1)$ rounds. Note that we do not hope to do better than $O(\log_s n)$ rounds as this is required even for computing the XOR of n bits.

Finally, note that the total communication is, *a fortiori*, $O(n)$ per round and $O(nR) = O(n)$ overall.

Streaming models. The above MPC model essentially resides in between two streaming models.

First, it is at least as strong as the "linear streaming" model, where one stores a (small) linear sketch of the input: if one has a linear sketch algorithm using space s and R passes, this immediately implies a parallel algorithm with local space s^2 (and $m = O(n/s^2)$ machines) and $O(R \log_s m)$ rounds.

Second, the above model can be simulated in the model of streaming *with a sorting primitive* [4]. The latter model is similar to the standard multi-pass streaming model, but allows for both annotating the stream with keys as we go through it and sorting the entire stream according to these keys. In particular, sorting is considered in this model to be just another pass. Then if we have a parallel algorithm with s space and R rounds, we also obtain a streaming-with-sorting algorithm with $O(s)$ space and $O(R)$ passes.

1.2 Our Results

In this work, we focus on graph problems for geometric graphs. We assume to have n points immersed in a low-dimensional space, such as $\mathbb{R}^2$ or a bounded doubling dimensional metric. Then we consider the complete graph on these points, where the weight of each edge is the distance between its endpoints.[2]

We give parallel algorithms for the following problems:

- Minimum Spanning Tree (MST): compute the minimum spanning tree on the nodes. Note that MST is related to the hierarchical agglomerative clustering

[1]Furthermore, it is hard to imagine a data set where $\sqrt[3]{n}$ is larger than the memory of a commodity machine.

[2]Since our algorithms work similarly for norms such as $\ell_1, \ell_2, \ell_\infty$, we are not specific about the norm.

with single linkage, a classic (and practical) clustering algorithm [40, 28].

We show how to compute a $(1+\epsilon)$-approximate MST over $\mathbb{R}^d$ in $\log_s^{O(1)} n$ rounds, as long as $(1/\epsilon)^{O(d)} < s$. Note that the number of rounds does not depend on ϵ or d. We extend the result to the case of a general point set with doubling dimension d. All our algorithms run in time $\tilde{O}(s) \cdot (1/\epsilon)^{O(d)}$ per machine per round.

We note that our algorithm outputs a complete tree (not just its cost as [18]). The tree consists of $n-1$ edges, which means that the output is also stored in a distributed manner. A sketch of the algorithm appears in Section 3.

- Earth-Mover Distance[3] (EMD): given an equipartition of the points into red and blue points, compute the min-cost red-blue matching. A generalization is the *transportation distance*, in which red and blue points have positive weights of the same total sum, and the goal is to find a min-cost matching between red and blue masses. EMD and its variants are a common metric in image vision [31, 21].

  We show how to approximate the EMD and transportation cost up to a factor of $1+\epsilon$ over $\mathbb{R}^2$ in $(\log_s n)^{O(1)}$ rounds, as long as $(\log n)^{(\epsilon^{-1} \log_s n)^{O(1)}} < s$. The running time per machine per round is polynomial in s. Note that, setting $s = 2^{\log^{1-c} n}$ for small enough $c > 0$, we obtain a sequential algorithm with overall running time of $n^{1+o(1)}$ for any fixed $\epsilon > 0$. Our algorithm can also be seen as an algorithm in the streaming-with-sorting model, achieving n^δ space and $1/\delta^{O(1)}$ rounds by setting $s = n^\delta$. Our algorithm does not output the actual matching (as [33] do).

All our algorithms fit into a general framework, termed Solve-And-Sketch, that we propose for such problems. The framework is naturally "parallelizable", and—we believe—is resilient to minor changes in the parallel model definition. We describe the general framework in Section 2, and place our algorithms within this framework. The actual implementation of the framework in the MPC model is deferred to the full version.

It is natural to ask whether our algorithms are optimal. Unfortunately, we do not know whether both approximation and small dimension are required for efficient algorithms. However, we show that if we could solve exact MST (cost) in $l_\infty^{O(\log n)}$, we could also solve sparse connectivity (in general graphs), for which we have indications of being impossible [9]. We also prove query-complexity lower bounds for MST in spaces with bounded doubling dimension in the black-box distance oracle model. In this setting, both approximation and dimension restriction are necessary. These results appear in the full version.

1.3 Motivation and Comparison to Previous Work

The model perspective. [27] have initiated the study of *dense* graph problems in the MapReduce model they define, showing constant-round algorithms for connected-components, MST, and other problems. In the dense setting, the parameters are such that $m \gg s \gg n$, where n is the number of vertices and m is the number of edges. In this case, the solution (the size of which is $O(n)$) fits on a single machine.

In this regime, the main technique is filtering (see also [30]), where one iteratively sparsifies the input until the entire problem fits on one machine, at which moment the problem is solved by a regular sequential algorithm. For example, for connected-components, one can just throw out edges locally, preserving the global connectivity, until the graph has size at most s.

Somewhat departing from this is the work of [15], who give algorithms for k-median and k-center, using $s = O(k^2 n^\delta)$. Instead of filtering, they employ (careful) sampling to reduce the size of the input until it fits in one machine and can be solved sequentially. Note that, while the entire "solution" is of size $n \gg s$, it can be represented by $k \ll s$ centers. [29] further generalize both the filtering and sampling approaches for certain greedy problems. In their case, the final solution of size $k \ll s$ is again computed on a single machine at the end.

Also highly relevant are the now-classic results on *coresets* [3, 16], which are a generic representation of (a subset of) input with the additional property of being mergeable. Corsets are often implementable in the MapReduce model (in fact, [15] can be seen as such an implementation). However, coresets have been mostly used for *geometric problems* (not graph problems), which often have a small solution representation.

We contrast the "dense" regime with the "sparse" regime, where s is much smaller than the size of the solution. Most notably, for the problem of computing the connected components in a sparse graph, we have no better algorithm than those following from the standard PRAM literature, despite a lot of attention from researchers. In fact, [9] suggest it may be hard to obtain a constant number of rounds for this problem.

Our algorithms rather fall in the "sparse" regime, as the solution (representation) is larger than the local space s. As such, it appears hard to apply filtering/sampling technique that drops part of the input from consideration. Indeed, our approach can rather be seen as a generalization of the notion of coreset.

We also mention other related works in MapReduce-like models, e.g., [12, 10, 7], which, however, require at least logarithmic parallel time.

The problems perspective. While we are not aware of a previous study of geometric graph problems in the MapReduce models, these problems have been studied extensively in other standard models, including 1) near-linear time algorithms, and 2) streaming algorithms.

Linear-time (approximate) algorithms for MST are now classic results [35, 11]. For EMD, it is only very recently that researchers found a near-linear time approximation algorithm [33] (following a line of work on the problem [36, 2, 38, 1, 25, 32]). Our framework naturally leads to near-linear time algorithms.

In the streaming model, a generic approach to approximating a large class of geometric graph problems has been introduced in [24]. The work of [24] has generally obtained logarithmic approximation for many problems and subsequently there has been a lot of research on improving these

[3]Also known as min-cost bichromatic matching, transportation distance, Wasserstein distance, and Kantorovich distance.

algorithms. Most relevantly, [18] have shown how to $(1+\epsilon)$-approximate the MST *cost*. We stress that their algorithm outputs the cost only and does not lead to an algorithm for computing the actual tree as we accomplish here.

Obtaining a $(1+\epsilon)$-approximation streaming algorithm for EMD is a well-known open question. The best known streaming algorithm obtains a $O(1/\delta)$ approximation in n^δ space for any $\delta > 0$ [5].

Our algorithmic framework immediately leads to an algorithm for computing $1+\epsilon$ approximation in n^δ space and $1/\delta^{O(1)}$ passes in the streaming-with-sorting model. In general, our EMD result implies one of the following: either 1) it illustrates the new parallel models as a concrete mid-point in the quest for an efficient streaming algorithm for EMD, or 2) it separates the new parallel models from the linear streaming model, showing them as practical models for sublinear space computation which are strictly more powerful than streaming. We do not know which of these cases is true, but either would be an interesting development in the area of sublinear algorithms.

1.4 Techniques

We now describe the main technical ideas behind our algorithms. Our MST algorithm is simple, but requires some careful analysis, while the EMD algorithm is technically the most involved and we describe the intuition behind it in the full version.

MST. To illustrate the main ideas involved in the algorithm it suffices to consider the problem over the 2D grid $[0,n]^2$. The framework consists of three conceptual parts: partition, local solution, and sketch. The partition is a standard quadtree decomposition, where we impose a hierarchical grid, randomly shifted in the space. In particular, each cell of the grid is recursively partitioned into $\sqrt{s} \times \sqrt{s}$ cells, until cell size is $\sqrt{s} \times \sqrt{s}$. The partition is naturally represented by a tree of arity s.

The other two parts are the crux of the algorithm. Consider first the following recursive naïve algorithm. Starting from leaves and going bottom-up, we compute the minimum spanning tree among the input points (local solution) at every cell in the quadtree, and then send a sketch of this tree to the upper-level cell. The problem is solved recursively in the upper-level cell by connecting partial trees obtained from the lower level.

However, such an algorithm does not yield a $(1+\epsilon)$-approximation. While constructing minimum spanning tree in a cell, the limited local view may force us to make an irrevocably bad decision. In particular, we may connect the nodes in the current cell, which in the optimal solution are connected by a path of nodes outside the cell (see the example in Figure 1.

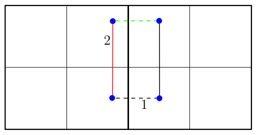

Figure 1: A bad example for the naïve MST algorithm.

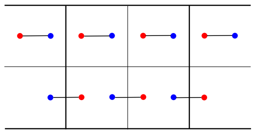

Figure 2: Local view is insufficient for EMD.

The challenge is to produce a local solution, without committing to decisions that may hurt in the future. To accomplish this, our local solution at a cell is to find the minimum spanning forest among the input points, using only *short edges*, of length at most ϵ times the diameter Δ of the cell. Note that it is possible that the local set of points remains disconnected.

Our sketch for each cell consists of an $\epsilon^2\Delta$-net[4] of points in the cell together with the information about connectivity between them in the current partial solution. Note that the size of the sketch is bounded by $O(\epsilon^{-4})$. This sketch is sent to the parent cell. Then the local solution at the parent node consists of constructing a minimum spanning forest for the connected components obtained from its children.

We also generalize our algorithm to the case of a point set with bounded doubling dimension. Here, the new challenge is to construct a good hierarchical partition first.

EMD. Our EMD algorithm adopts the general principle from MST, though the "solution" and "sketch" concepts become more intricate. Consider the case of EMD over $[n]^2$. As in MST, we partition the space using a hierarchical grid, represented by a tree with arity s.

In contrast to the MST algorithm, there are no local "safe" decisions one can make whatsoever. Consider Figure 2. The two rows of points are identical according to the local view. However, in one case we should match all points internally, and in the other, we should leave the end points to be matched outside. As far as the local view is concerned, either partial solution may be the right one. If we locally commit to the wrong one, we are not able to achieve a $1+\epsilon$ approximation no matter what we do next. Therefore, we need to sketch the *entire set of local solutions*.

This is exactly what we accomplish: we sketch the set of *all* possible local solutions. While reminiscent of the dynamic programming philosophy, our case is burdened by the requirement that the representation use sublinear (local) space. Our algorithm manages to sketch this set of relevant local solutions approximately. Suppose we define a function F of d coordinates, one for "each position" in the local cell. In particular, F takes as argument a vector $x \in \mathbb{Z}^d$ that specifies, for each $i \in [d]$, how many points of each color are left unmatched at position i. Then we can define $F(x)$ to be the cost of the optimal matching of the rest of the points (with points specified by x excluded from the local matching).

Ideally, we would like to sketch the function $F : \mathbb{Z}^d \to \mathbb{R}_+$. Unfortunately, even for a monotone, convex, and Lipschitz function F with $d = 2$, a sketch is generally not possible. (In our case, F is not even monotone.) What we show instead is that we can sketch the function $F'(x) = F(x) + \|x\|_1 \cdot A$, for some convenient factor A. The additional term of $\|x\|_1 \cdot A$ is tolerable as it captures part of the matching cost *at the higher level*. As a result, our sketch of F' consists of $F'(x)$ values at $(\epsilon^{-1}\log n)^{O(d)}$ well-chosen values of x.

These ideas eventually lead to an *information-theoretic* algorithm for EMD, namely with the promised guarantees on space, communication, and rounds. It remains to make the *running time* of the local step polynomial in s. To solve this problem, for each j we find a convex function F''_j which is *consistent with the sketch* of F'_j. Using the convexity of F'_j and the guarantees of the sketch, we can show that F''_j

[4]An r-net of a point set is the maximal subset with pairwise distances at least r.

approximates F'_j pointwise. We then obtain a convex optimization problem (in our case we in fact reduce it to a linear program), which can be solved in time polynomial in s.

2. PRELIMINARIES: SOLVE-AND-SKETCH FRAMEWORK

We now introduce the framework for our algorithms, termed Solve-And-Sketch. Its main purpose is to identify and decouple the crux of the algorithm for the specific problem from the implementation of the algorithm in the parallel model such as MPC.

The framework requires a "nice" hierarchical partition of the space. We view the hierarchical partition as a tree, where the arity is upper bounded by $\sqrt{s}$, and the depth is $O(\log_s n)$. The actual computation is broken down into small "local computation chunks," arranged according to the hierarchical partition. The computation proceeds bottom-up, where at each node, the input (from below) is processed and the results are compressed into a small *sketch* that is sent up to the parent. Each level of the tree is processed in parallel, with each node assigned to a machine.

In particular, the local computation at a node, termed "unit step," consists of two steps:

Solve: Given the local inputs, we compute the set of partial or potential solutions. For leaves, the local information consists of the points in that part, and for internal nodes, it is the information obtained from the children.

Sketch: Sketch the partial solution(s), using total space at most $p_u \leq \sqrt{s}$, and send this up the tree to the parent as a representation of solution(s) in this part.

The main challenges are how to: 1) compute the partition, 2) define the right concept of a "local solution" in a part, and 3) sketch this concept as a sufficient representation of all potential solutions in this part. Often the naïve choice of the a local solution cannot be used, because it either ignores global information in a way that can damage the optimality of the algorithm, or it cannot be represented in sublinear space.

We now define more formally the notions of partition and of a unit step.

Hierarchical Partition. We use a hierarchical partition for inputs in $(\mathbb{R}^d, \ell_2)$ that is an analogue of a randomly-shifted quad-tree but with a *higher branching factor* than the usual 2^d. We denote the branching factor by c. We describe this partitioning scheme next (see the full version for additional details). The partition we use to compute MST in a low doubling dimension metric space is more involved: see the full version for the construction.

We assume that the input points have integer coordinates in the range $[0, \Delta]$, where $\Delta = n^{O(1)}$. We show how to remove this bounded aspect ratio assumption in the full version for the problems we consider. We construct a *randomized hierarchical partition* $\mathcal{P}$ (i.e., a distribution over hierarchical partitions). A fixed partition sampled from $\mathcal{P}$ is denoted $P = (P_0, \ldots, P_L)$, where P_ℓ is a subdivision of $P_{\ell-1}$. Let $v \in \mathbb{R}^d$ be a vector chosen uniformly at random from $(-\Delta, 0]^d$; P is entirely determined by the choice of v. The top level P_L has a single part containing the whole input, and is identified with the cube $\{x : \forall i \; v_i \leq x_i \leq v_i + 2\Delta\}$. Then we construct $P_{\ell-1}$ from P_ℓ by subdividing each cube associated with a part in P_ℓ into c equal sized cubes (via a grid with side-length $c^{1/d}$), thus creating a part associated with each smaller cube. In the final level P_0, each part is a singleton, i.e., all associated cubes contain at most a single point from the input. Since we assumed all points have integer coordinates in $[0, \Delta]$, it is enough to take $L = d \log_c \Delta = O(d \log_c n)$. We refer to the parts of each partition P_ℓ as *cells*. For a level-ℓ cell $C \in P_\ell$, we define the *child cells* of C to be those cells $C' \in P_{\ell-1}$ that subdivide C, i.e., $C' \subseteq C$. For our implementation, we also need to label each child cell of C with an integer in $[c]$.

Unit Step. The other important component of the sketch and solve framework is the unit step, which is an algorithm $\mathcal{A}_u$ that is applied to each cell $C \in P_\ell$ for $\ell = 1, \ldots, L$. At level 1, $\mathcal{A}_u$ takes as input the points in C, and at level $\ell > 1$, $\mathcal{A}_u$ takes as input the union of outputs of the unit steps applied to the children of C. The output of $\mathcal{A}_u$ on the top-most cell P_L is the output of the problem (perhaps after some post-processing). We define functions p_u, t_u, s_u as follows: on input of size n_u, $\mathcal{A}_u$ produces an output of size at most $p_u(n_u)$, runs in time at most $t_u(n_u)$, and uses total space $s_u(n_u)$. We require that, on empty input, $\mathcal{A}_u$ produces empty output. We call the algorithm that applies $\mathcal{A}_u$ to each cell of the partition in the above fashion the *Solve-And-Sketch* algorithm.

We prove that once we have a unit step algorithm for a problem, we also obtain a complete parallel algorithm for the said problem. Hence designing the unit step for a problem is the crux for obtaining a parallel algorithm and is decoupled from the actual implementation specifics in the considered parallel model.

THEOREM 2.1 (SOLVE-AND-SKETCH). *Fix space parameter $s = (\log n)^{\Omega(d)}$ of the MPC model. Suppose there is a unit step algorithm using local time $t_u(n_u)$, space $s_u(n_u)$, and output size $p_u(n_u)$ on input of size n_u. Assume the functions t_u, s_u, p_u are non-decreasing, and also satisfy:*

$$s_u(p_u(s)) \leq s^{1/3} \quad \text{and} \quad p_u(s) \leq s^{1/3}.$$

Then we can set $c = s^{\Theta(1)}$ and $L = O(\log_s n)$ in the partitioning from above, and we can implement the resulting Solve-And-Sketch algorithm in the MPC model in $(\log_s n)^{O(1)}$ rounds. Local runtime is $s \cdot t_u(s) \cdot (\log n)^{O(1)}$ (per machine per round).

We defer the proof of the theorem, along with other details of implementation in the MPC model, to the full version. In the remainder of this extended abstract, we are concerned with designing unit step algorithms with the requisite parameters for the two problems we consider. By Theorem 2.1, this suffices for an efficient MPC algorithm. In the full version, we also show that, using a different parametrization for the hierarchical partition, out unit step algorithms also imply near-linear time sequential algorithms.

3. MINIMUM SPANNING TREE

In this section we prove the existence of an efficient MPC algorithm that computes a spanning tree of a given point set in Euclidean space of approximately minimal cost.

THEOREM 3.1. *Let $\epsilon > 0$, and $s \geq (\epsilon^{-1} \log_s n)^{O(1)}$. Then there exists an MPC algorithm that, on an input set S in $\mathbb{R}^d$ (where $|S| = n$), runs in $(\log_s n)^{O(1)}$ rounds and outputs a* *spanning tree of cost (under the Euclidean distance metric ℓ_2^d for $d = O(1)$) at most $1+\epsilon$ factor larger than the optimal. Moreover, the running time per machine is near linear in the input size n_u, namely $O(n_u \epsilon^{-d} \log^{O(1)} n_u)$.*

We prove the theorem above by exhibiting a unit step algorithm within the Solve-And-Sketch framework from Section 2. Our unit step algorithm works with partitions more general than the quadtree-based partition described in Section 2. This allows us to apply the unit step to point sets in low doubling dimension as well, once we have constructed an appropriate hierarchical partition. See the full version for more details on the doubling dimension case. For the remainder of this section, we focus on the low-dimensional Euclidean case for simplicity.

3.1 Hierarchical Partitions

In this section we define some additional notation related to hierarchical partitions. We also state the properties of the hierarchical partitions of Euclidean space defined in Section 2 which are necessary for our analysis. In the full version we abstract the concept of partitions with these properties.

We denote the unique cell at level ℓ containing a point x as $C_\ell(x)$, i.e., $C_\ell(x)$ is defined by $x \in C_\ell(x)$ and $C_\ell(x) \in P_\ell$. For $\ell' \leq \ell$ and $C \in P_{\ell'}$, we define $C_\ell(C)$ analogously as the unique cell in the level ℓ containing C.

We use the notation $\rho(x, y)$ for the Euclidean distance between x and y, and $\Delta(S) = \max_{x,y \in S} \rho(x, y)$ for the diameter of a set S. The *diameter at level ℓ* of a partition $P = (P_0, \dots, P_L)$ is denoted $\Delta(P_\ell) = \max_{C \in P_\ell} \Delta(C)$. Let us define $\Delta_\ell = c^{(\ell-L)/d} 2\sqrt{d}\Delta$, where c is the branching factor of the partition. The two properties of the randomized partition $\mathcal{P}$ defined in Section 2 that are essential to our analysis are the following:

1. (Bounded diameter) For every deterministic partition $P = (P_0, \dots, P_L)$ in the support of $\mathcal{P}$, and for all $\ell \in \{0, \dots, L\}$, $\Delta(P_\ell) \leq \Delta_\ell$.

2. (Probability of cutting an edge) For every $x, y \in [0, \Delta]$, and for all $\ell \in \{0, \dots, L\}$,
$$\Pr[C_\ell(x) \neq C_\ell(y)] \leq O(d) \frac{\rho(x, y)}{\Delta_\ell},$$
where the probability is taken over the choice of a deterministic hierarchical partition from $\mathcal{P}$.

For a proof of the above properties, see the full version, and also [6].

3.2 The Unit Step Algorithm

Recall that a Solve-and-Sketch (SAS) samples a hierarchical partition $P = (P_0, \dots, P_L)$ of the input, and proceeds through L levels. In level ℓ, a *unit step* algorithm is executed in each cell C of the partition P_ℓ, with input the union of the outputs of the unit steps applied to the children of C. Our MST unit step also outputs a subset of the edges of a spanning tree in addition to the input for the next level. In particular, the unit step computes a minimum spanning forest of the (possibly disconnected) subgraph consisting of edges between points in the cell of length at most an $\epsilon\Delta_\ell$. By not including longer edges we ensure that ignoring the edges that cross cell boundaries does not cost us a constant factor in the quality of the approximation (see Figure 1). The edges of the computed minimum spanning forest are output as a part of the constructed spanning tree. For the next level we output an $\epsilon^2\Delta_\ell$-covering of points in the cell, annotated by the connected components of the minimum spanning forest. In a space of constant dimension we can construct such a covering of size $\epsilon^{-O(1)}$. The reason why the distance information given by the covering is accurate enough for our approximation is that all edges between different connected components in the spanning forest constructed so far are either long or have been crossing in the previous level.

We describe the unit step as Algorithm 1. Then Theorem 3.1 follows from Theorem 2.1 and the guarantees on space and time complexity, as well as the approximation guarantees of Algorithm 1.

```
Algorithm 1: Unit Step at Level $\ell$
input : Cell $C \in P_\ell$; a collection $V(C)$ of points in $C$,
        and a partition $Q = \{Q_1, \dots Q_k\}$ of $V(C)$ into
        previously computed connected components.
1 $\theta := 0$
2 while $k > 1$ and $\theta \leq \epsilon\Delta_\ell$ do
3     $\tau := \min_{i,j, i \neq j} \min_{u \in Q_i, v \in Q_j} \rho(u, v)$
4     Find $u \in Q_i$ and $v \in Q_j$ for some $i$ and $j$ such that
      $i \neq j$ and $\rho(u, v) \leq (1+\epsilon)\tau$.
5     $\theta := \rho(u, v)$
6     if $\theta \leq \epsilon\Delta_\ell$ then
7         Output tree edge $(u, v)$.
8         Merge $Q_i$ and $Q_j$ and update $Q$ and $k$.
output: $V' \subseteq V$, an $\epsilon^2\Delta_\ell$-covering for $C$, the partition
        $Q(V')$ induced by $Q$ on $V'$.
```

Notice that Algorithm 1 implements a variant of Kruskal's algorithm, with the caveats that we ignore edges longer than $\epsilon\Delta_\ell$ as well as edges crossing the boundary of C, and that we also join only the approximately closest pair of connected components, rather than the closest pair. This last choice enables the application of efficient algorithms for approximate nearest neighbor search [23, 22], which are used to identify which connected components to connect. Consequently, we achieve near-linear total running time.

Let T^* be some optimum minimum spanning tree. For a tree T, let $\rho(T)$ denote the cost of the tree $\sum_{(u,v) \in T} \rho(u, v)$. The following theorem is our main approximation result for the SAS algorithm with unit step Algorithm 1.

THEOREM 3.2. *For $\epsilon \leq \frac{1}{4}$ and $c \geq 2^d$, the spanning tree T° output by the Solve-and-Sketch algorithm with partition P sampled from $\mathcal{P}$ and unit step Algorithm 1 satisfies:*
$$\mathop{\mathbb{E}}_{P \sim \mathcal{P}} [\rho(T^\circ)] \leq (1 + \epsilon O(Ld))\rho(T^*).$$

It is natural to attempt to prove Theorem 3.2 by relating the SAS algorithm with unit step Algorithm 1 to a known MST algorithm, e.g., Kruskal's algorithm (which our algorithm most closely resembles). There are several difficulties, arising from approximations that we use in order to achieve efficiency in terms of communication, running time, and space. For example, our algorithm only keeps progressively coarser coverings of the input between phases, and thus does not have exact information about distances between connected components. Nevertheless, it is known that an approximate implementation of Kruskal's algorithm still outputs an approximate MST [23, Section 3.3.1]. However, our setting presents a further difficulty: because we work in a parallel environment, Algorithm 1 completely ignores any edges crossing the boundary of the cell it is currently applied to. Such edges could have small length, which makes it generally impossible to show that our algorithm implements Kruskal's algorithm even approximately for the complete graph with edge weights given by the Euclidean metric. Instead, we are able to relate our algorithm to a run of Kruskal's algorithm on the complete graph with *modified edge weights* $w_P : S \times S \to \mathbb{R}_+$. These weights are a function of the hierarchical partition P; they are always an upper bound on the true distance ρ, and give larger weight to edges that cross the boundaries of P_ℓ for larger ℓ . We are able to show (Lemma 3.13) that the length (under ρ) of the i-th edge output by (a sequential simulation) of our algorithm is at most a factor $1+\epsilon$ larger than the weight (under w_P) of the i-the edge output by Kruskal's algorithm, when run on the complete graph with edge weights w_P. The proof is then completed by arguing that for each $u, v \in S$, $w_P(u,v)$ approximates $\rho(u,v)$ in expectation when P is sampled from a distance preserving partition (Lemma 3.4).

We define the following types of edges based on the position of their endpoints with respect to the space partition used by the algorithm.

Definition 3.3 (Crossing and non-crossing edges). *An edge (u,v) is* crossing *in level ℓ if $C_\ell(u) \neq C_\ell(v)$ and* non-crossing *otherwise.*

Also for each edge we define the *crossing* level, which is useful in the analysis. For an edge (u,v) let its *crossing level* $\ell_c(u,v)$ be the largest integer such that $C_{\ell_c(u,v)}(u) \neq C_{\ell_c(u,v)}(v)$. Let $\mathcal{P}$ be a randomized (a,b)-partition of $M(S,\rho)$ with L levels. For every deterministic partition P in the support of $\mathcal{P}$, we define the *modified weights* $w_P(u,v) = \rho(u,v) + \epsilon\Delta_{\ell_c(u,v)}$.

We show that the modified weights $w_P(u,v)$ approximate the original distances $\rho(u,v)$ in expectation. This lemma and its proof are similar to arguments used in recent work on approximating the Earth-Mover Distance in near-linear time [33], and date back to Arora's work on approximation algorithms for the Euclidean Traveling Salesman Problem [6]. The proof appears in the full version.

Lemma 3.4. *For all $u, v \in S$, $\rho(u,v) \le \mathbb{E}_{P\sim\mathcal{P}}[w_P(u,v)] \le (1 + O(\epsilon L d))\rho(u,v)$.*

Recall that in Algorithm 1 for a cell $C \in P_\ell$ the set $V(C)$ is a subset of points of C considered at level ℓ. Also recall that $C_\ell(u)$ is the cell containing u at level ℓ. We use the following notation to denote the closest neighbor of u considered at level ℓ.

Definition 3.5. *For $u \in S$ let $N_\ell(u)$ be the nearest neighbor to u in $V(C_\ell(u)) \cap C_{\ell-1}(u)$, i.e.,*

$$N_\ell(u) = \arg\min_{v \in V(C_\ell(u)) \cap C_{\ell-1}(u)} \rho(u,v).$$

For two points u and v, we use the following distance measure $\rho_\ell(u,v)$ in the analysis.

Definition 3.6. *For an edge (u,v) we define $\rho_\ell(u,v) = \rho(N_\ell(u), N_\ell(v))$ to be the distance between the nearest neighbors of u and v at level ℓ.*

The next lemma (with a proof omitted from this extended abstract) shows that ρ_ℓ is an approximation to ρ.

Lemma 3.7. *For every $C \in P_\ell$ and $u, v \in C$, it holds that $|\rho_\ell(u,v) - \rho(u,v)| \le 2\epsilon^2\Delta_{\ell-1}$.*

To complete our analysis we need to further characterize edges according to their status during the execution of the algorithm.

Definition 3.8 (Short and long edges). *We call an edge (u,v)* short *in level ℓ if $\rho_\ell(u,v) \le \frac{\epsilon}{1+\epsilon}\Delta_\ell$, and long otherwise.*

Definition 3.9 (Processing level and sequence). *For an edge e in T°, the* processing level *$\ell_p(e)$ is the integer ℓ such that e is output by the unit step applied to some $C \in P_\ell$. Consider a sequential simulation of the SAS algorithm with unit step Algorithm 1, in which at each level ℓ, the unit step is applied to each cell $C \in P_\ell$ sequentially in an arbitrary order. The* processing sequence *$(e_1, \ldots, e_{n-1})$ consists of the edges of T° in the order in which they are output by the above sequential simulation.*

Definition 3.10 (Intercluster edges). *The* forest at step i, *denoted T_i°, is defined as the forest $\{e_1, \ldots, e_i\}$. An edge $e = (u,v)$ is* intercluster *at step i if u and v lie in different connected components of T_{i-1}°. We denote the set of all intercluster edges at step i as I_i.*

Lemma 3.11. *Let $\epsilon \le \frac{1}{4}$ and $a \le \frac{1}{2}$. For every vertex u, level ℓ and step $i > 1$ such that $\ell_p(e_i) \ge \ell$ the vertices u and $N_\ell(u)$ are in the same connected component of T_{i-1}°.*

Proof. Note that it suffices to prove the claim for the smallest i such that $\ell_p(e_i) \ge \ell$. Fix such i. Assume for contradiction that for some ℓ the vertices u and $N_\ell(u)$ are in different connected components of T_{i-1}°. Fix the smallest such ℓ. If $\ell = 1$, then $N_\ell(u) = u$, so we may assume $\ell \ge 2$. Let $C = C_\ell(u)$ and $C' = C_{\ell-1}(u)$. At the end of the execution of Algorithm 1 in cell C', the partition Q of $V(C')$ into connected components is a subdivision of the connected components of T_{i-1}° restricted to $V(C')$. By the choice of ℓ, u and $N_{\ell-1}(u)$ are in the same connected component of T_{i-1}°, and, since we assumed that u and $N_\ell(u)$ are in different connected components of T_{i-1}°, it must be the case that $N_{\ell-1}(u)$ and $N_\ell(u)$ are in different connected components in Q, i.e., $N_{\ell-1}(u) \in Q_k$ and $N_\ell(u) \in Q_{k'}$ for $k \neq k'$. Since $V(C) \cap C'$ is a $\epsilon^2\Delta_{\ell-1}$-covering of C' and $V(C')$ is a $\epsilon^2\Delta_{\ell-2}$-covering of C', we have $\rho(u, N_\ell(u)) \le \epsilon^2\Delta_{\ell-1}$ and $\rho(u, N_{\ell-1}(u)) \le \epsilon^2\Delta_{\ell-2}$. Then, by the triangle inequality, $\tau \le \rho(N_{\ell-1}(u), N_\ell(u)) \le (1+a)\epsilon^2\Delta_{\ell-1}$, and the algorithm finds $u' \in Q_k, v' \in Q_{k'}$ such that $\theta = \rho(u',v') \le (1+\epsilon)\tau \le \epsilon^2(1+\epsilon)(1+a)\Delta_{\ell-1}$. Since for $\epsilon \le \frac{1}{4}$ and $a \le \frac{1}{2}$, $\epsilon(1+\epsilon)(1+a) < 1$, this contradicts the termination condition for the main loop of Algorithm 1. ∎

Lemma 3.13 is the key part of the proof of Theorem 3.2. It shows that the cost of the i-th edge output by our algorithm is bounded in terms of the cost of i-th edge output by Kruskal's algorithm. In this extended abstract we omit the full proof and give a sketch instead.

Definition 3.12 (Kruskal's edge). *Let $e_i^{w_P}$ be the i-th edge output by Kruskal's algorithm when run on the complete graph on S with edge weights $w_P : S \times S \to \mathbb{R}$.*

LEMMA 3.13. *If $\epsilon \le \frac{1}{4}$ and $a \le \frac{1}{2}$, then for each i it holds that $\rho(e_i) \le (1+O(\epsilon))w_P(e_i^{w_P})$.*

PROOF SKETCH. We denote the shortest intercluster edge at step i as $e_i^+ = \arg\min_{e \in I_i} w_P(e)$.

First we show that the weight of $e_i^{w_P}$ is bounded by the weight of e_i^+ in Proposition 3.14. This argument is due to Indyk [23, Section 3.3.1, Lemma 11].

PROPOSITION 3.14. *For each i it holds that $w_P(e_i^+) \le w_P(e_i^{w_P})$.*

PROOF. Note that T_{i-1}° has $n-i+1$ connected components. Because $\{e_1^{w_P}, \ldots, e_i^{w_P}\}$ is a forest, there exists $j \le i$ such the endpoints of $e_j^{w_P}$ lie in different connected components of T_{i-1}°. Thus, by definition of e^+ we have $w_P(e_i^+) \le w_P(e_j^{w_P})$. Because the edges output by Kruskal's algorithm satisfy that $w_P(e_j^{w_P}) \le w_P(e_i^{w_P})$ for $j \le i$, the lemma follows. ∎

Using Proposition 3.14 it suffices to show that $\rho(e_i) \le (1+O(\epsilon))w_P(e_i^+)$ to complete the proof. The proof considers three cases: **(I)** $\ell_c(e_i^+) \ge \ell_p(e_i)$, **(II)** $\ell_c(e_i^+) = \ell_p(e_i)-1$, and **(III)** $\ell_c(e_i^+) < \ell_p(e_i)-1$. Case (I) is relatively easy, and relies on the fact that in this case $w_P(e_i^+)$ is large by definition: the proof is omitted from this sketch. Cases (II) and (III) use the following proposition, proved in the full version of the paper. It shows that the i-th edge e_i output by (the sequential simulation) of the SAS algorithm is approximately the shortest non-crossing intercluster edge.

PROPOSITION 3.15. *Let $\epsilon \le \frac{1}{4}$ and $c \ge 2^d$. If $e \in I_i$ is non-crossing at level $\ell_p(e_i)$ then $\rho(e_i) \le (1+\epsilon)\rho_{\ell_p(e_i)}(e)$.*

Case (II) follows from Proposition 3.15 and definitions by a straightforward argument, which we omit. The proof of case (III) is the most delicate, and we give the proof below. We first state another proposition, which is proved in the full version.

PROPOSITION 3.16. *Let $\epsilon \le \frac{1}{4}$ and $c \ge 2^d$. Every $e \in I_i$ is either crossing or long in level $\ell_p(e_i)-1$.*

In case (III), by Proposition 3.16, since e_i^+ was not crossing in level $\ell_p(e_i)-1$, then e_i^+ must have been long. Thus,

$$\begin{aligned}
\rho(e_i^+) &\ge \rho_{\ell_p(e_i-1)}(e_i^+) - \epsilon^2\Delta_{\ell_p(e_i)-2} \\
&> \frac{\epsilon}{1+\epsilon}\Delta_{\ell_p(e_i)-1} - \epsilon^2\Delta_{\ell_p(e_i)-2} \\
&= \left(\frac{1}{1+\epsilon} - c^{-1/d}\epsilon\right)\epsilon\Delta_{\ell_p(e_i)-1} \\
&> (1-(1+a)\epsilon)\epsilon\Delta_{\ell_p(e_i)-1},
\end{aligned}$$

where the first inequality holds by Lemma 3.7, the second inequality holds by definition of a long edge at level $\ell_p(e_i)-1$ (Definition 3.8) and the third equality holds because by definition $\Delta_{\ell_p(e_i)-2} = c^{-1/d}\Delta_{\ell_p(e_i)-1}$.

Then we have:

$$\begin{aligned}
\rho(e_i) &\le (1+\epsilon)\rho_{\ell_p(e_i)}(e_i^+) \\
&\le (1+\epsilon)\rho(e_i^+) + (1+\epsilon)\epsilon^2\Delta_{\ell_p(e_i)-1} \\
&\le \left(1 + \left(1 + \frac{1+\epsilon}{1-(1+c^{-1/d})\epsilon}\right)\epsilon\right)\rho(e_i^+) \\
&\le (1+O(\epsilon))w_P(e_i^+),
\end{aligned}$$

where the first inequality holds by Proposition 3.15, the second holds by 3.7, the third uses the calculation above and the last one is a direct calculation. ∎

Theorem 3.2 follows from Lemma 3.13 and Lemma 3.4 using standard arguments. We omit the proof from this extended abstract.

3.3 Proof of Theorem 3.1

Theorem 3.1 follows from Theorem 2.1, Theorem 3.2, and the following lemma (with a proof omitted), which gives guarantees on the time and space complexity of Algorithm 1.

LEMMA 3.17. *The unit step Algorithm 1 has space complexity $s_u(n_u) = n_u \log^{O(1)} n_u$ words, and time complexity $t_u(n_u) = \epsilon^{-d} n_u \log^{O(1)} n_u$. Moreover, the output size is $p_u = O(\epsilon^{-d})$ words.*

4. THE TRANSPORTATION PROBLEM

In this section we sketch the parallel algorithms for computing the cost of the Earth-Mover Distance and Transportation problems.

In the Transportation problem we are given two sets of points A, B in a metric space (M, ρ), and a demand function $\psi : A \cup B \to \mathbb{N}$ such that $\sum_{u \in A}\psi(u) = \sum_{v \in B}\psi(v)$. The Transportation cost between A and B given demands ψ is the value of the minimum cost flow from A to B such that the demands are satisfied (i.e., the flow out of each point $u \in A$ is $\psi(u)$ and the flow into each point $v \in B$ is $\psi(v)$, and the costs are given by the metric ρ. We denote the transportation cost, i.e., the minimum cost of a feasible flow, by $\text{cost}_\rho(A, B, \psi)$. When for all $u \in A, v \in B$, $\psi(u) = \psi(v) = 1$, the minimal flow forms a matching, and, therefore, its value is just the minimum cost of a perfect bichromatic matching. In this case $\text{cost}_\rho(A, B, \psi)$ is the Earth-Mover Distance between A and B, and we denote it simply by $\text{cost}_\rho(A, B)$.

In this paper we are concerned with the Euclidean Transportation cost problem, in which we assume that A and B are sets of points in the plain $\mathbb{R}^2$, and ρ is the usuall Euclidean distance ℓ_2^2. Therefore, for the rest of the section we assume that ρ is the Euclidean distance metric, and we write $\text{cost}(A, B, \psi)$ for $\text{cost}_\rho(A, B, \psi)$. Our results generalize to any norm on $\mathbb{R}^d$ for $d = O(1)$, but we focus on the Euclidean case for simplicity.

The main result of this section is the following theorem.

THEOREM 4.1 (TRANSPORTATION COST PROBLEM). *Let $\epsilon > 0$, space $s \ge (\log n)^{(\epsilon^{-1}\log_s n)^{\Omega(1)}}$, and max demand be $U = n^{O(1)}$. Then there exists an MPC algorithm with space parameter s that, on input sets $A, B \subseteq \mathbb{R}^2$, $|A|+|B| = n$, and demand function $\psi : A \cup B \to [0, U]$ such that $\sum_{u \in A}\psi(u) = \sum_{b \in B}\psi(v)$, runs in $(\log_s n)^{O(1)}$ rounds and outputs a $(1+\epsilon)$-approximation to $\text{cost}(A, B, \psi)$. Moreover, the local running time per machine (per round) is polynomial in s.*

Our methods also imply a *near-linear time sequential* algorithm for the Transportation cost problem, answering an open problem from [33].

THEOREM 4.2 (NEAR-LINEAR TIME). *Let $\epsilon > 0$ and $U = n^{O(1)}$. There exists an algorithm with running time $n^{1+o_\epsilon(1)}$ that, on input sets $A, B \subseteq \mathbb{R}^2$, $|A| + |B| = n$, and* *demand function* $\psi : A \cup B \to [0, U]$ *such that* $\sum_{a \in A_\psi} \psi(a) = \sum_{b \in B_\psi} \psi(b)$*, outputs a* $(1+\epsilon)$*-approximation to* $\mathrm{cost}(A, B, \psi)$.

We develop the proof in four steps. First, we impose a hierarchical partition that also defines a modified distance metric ρ_g that approximates ρ in expectation (our only step similar to one from [33]). The new metric has a tree structure that allows us to develop a recursive optimality condition for the Transportation problem that uses only local information and can be sketched in small space. Second, we define a generalized cost function F, which captures all the local solutions of a corresponding part/node in the tree. Third, we develop an "information theoretic" parallel algorithm; the algorithm does not run in polynomial time but obeys the space and communication constraints of our model. Finally, we modify the information theoretic algorithm to obtain a time-efficient algorithm.

New distance. As with MST, we sample randomized hierarchical partition $P = (P_1, \ldots, P_L)$ of $A \cup B$ using a quadtree with branching factor c, as described in Section 2. For each level ℓ of the grid, define Δ_ℓ to be the side-length of cells at that level: $\Delta_\ell = \Delta/(\sqrt{c})^{L-\ell}$. At level ℓ, we also consider a subgrid of squares of side length $\delta\Delta_\ell$ imposed over P_ℓ. We call the centers of the subgrid a *net at level* ℓ.

We define a new *grid distance* ρ_g based on the hierarchical partition (i.e., $\rho_g(u, v)$ is a random variable, and is entirely determined by P). The construction of ρ_g is similar to the definition of the weights w_P in the analysis of our MST algorithm, and also to the metric defined in [33]. The two main properties of ρ_g are that **(1)** for any u, v that lie in different cells of P_ℓ, $\rho_g(u, v) \ge \delta\Delta_\ell$, and **(2)** for any input A, B, ψ, with probability 9/10, $\mathrm{cost}(A, B, \psi) \le \mathrm{cost}_{\rho_g}(A, B, \psi)$.

We defer the precise definition of ρ_g and the proofs of the above two statements to the full version of the paper.

Cost Function. For a given grid cell C at level ℓ, we define a multi-argument function F that represents the cost of solutions in the cell C. The number of arguments of F is $d = 1/\delta^2 - 1$. In particular, there is one argument, x_r, corresponding to each point r from from the grid $\mathcal{N} = \mathcal{N}_\ell \cap C$, except exactly one (arbitrarily chosen) called r^*.

Let $A_C = A \cap C$ and $B_C = B \cap C$, and, for each point $r \in \mathcal{N} \cap C$, let A_r be the set of points in A_C for which r is the nearest neighbor in $\mathcal{N}$, and define B_r analogously.

The function F on a vector $x \in \mathbb{R}^d$ is the value of a solution of a min-cost flow problem. We define a network by taking the complete bipartite graph on A_C and B_C, with costs given by the grid distance ρ_g. We also connect each net point r to A_r; finally we connect r^* to all other netpoints. The interpretation of the arguments of F is that $x_r > 0$ means x_r demand from A_r has to flow to points outside C through r and $x_r < 0$ means that $-x_r$ demand from B_r has to flow in from from points outside C through r. Having specified all values of x_r for $r \in \mathcal{N}^\circ$, we route the imbalance through r^*. The cost for flow coming from the outside is $\delta\Delta_\ell/2$: by the definition of the grid distance this is a valid lower bound on the final cost. Modulo these constraints, and the demands specified by ψ on A_C and B_C, $F(x)$ gives the minimum cost of a flow from A_C to B_C. The details of the definition of F appear in the full version of the paper.

The crucial property of F is that it is *sketchable*: there exists a small size data structure which allows us to approximate $F(x)$ for any x. The main ingredient in the construction of the sketch is the following lemma.

LEMMA 4.3. *Fix* $\epsilon > 0$ *and suppose* $\delta = \epsilon^{O(1)}$. *There exists an* $\epsilon' = \epsilon^{O(1)}$ *such that for all feasible* $x, x' \in \mathbb{R}^{\mathcal{N}^\circ}$ *satisfying* $\forall i : x'_i/x_i \in [1-\epsilon', 1+\epsilon']$, *we have* $|F(x) - F(x')| \le \epsilon F(x)$.

The proof of Lemma 4.3 (omitted here) relies on the following two facts

- *(Lower bound)* $F(x) \ge \|x\|_1 \delta\Delta_\ell/2$ (because the cost of a unit of flow coming from outside is $\delta\Delta_\ell/2$;
- *(Lipschitz continuity)* for any coordinate $r \in \mathcal{N}$, we have $|F(x + \alpha e_r) - F(x)| \le O(\alpha\Delta_\ell)$, where e_r is standard basis vector.

Given Lemma 4.3, the construction of a sketch for F is simple: we record the value of F at at all x for which each coordinate is an integral power of ϵ'. This gives a sketch of size $(\log n)^{1/\epsilon^{O(1)}}$.

The Algorithm. Our algorithm for the transportation cost problem is based on a recursive formulation. We characterize $F(x)$ for any cell C with children $C_1, \ldots, C_c$ as an optimization problem whose objective function has two components: the sum of the cost functions $F_i(x^i)$ for the child cells C_i, and the cost of routing flow between the relevant netpoints. Evaluating $F(0)$ for the unique cell $C \in P_L$ that contains the entire input gives the transportation cost. This recursive definition, together with the fact that each F_i can be approximated from a small size sketch, is sufficient for us to define a *unit step algorithm* satisfying the required space and output constraints. However, the naïve unit step algorithm does not run in polynomial time, because it must iterate through all possible parameter vectors x^i for the cost functions F_i for each child cell C_i. We address this issue by computing a convex function $\tilde{F}_i$ that gives at least as good an approximation to F_i as the sketch of F_i. In particular we let $\tilde{F}_i$ be the largest convex function which agrees with the sketch of each F_i. Because F_i is a convex function itself, and the sketch is just the list of values of $F_i(x)$ for a small set of parameter vectors x, $\tilde{F}_i$ must be sandwiched between the estimate given by the sketch and the actual F_i. This gives the approximation guarantee.

The functions $\tilde{F}_i$ are "piecewise linear" and can easily be absorbed into a larger linear program (LP) to compute F for a cell C. In the end, the local running time is polynomial in s, because the resulting LPs can be solved with an arbitrary polynomial time LP algorithm.

5. REFERENCES

[1] P. Agarwal and K. Varadarajan. A near-linear constant factor approximation for Euclidean matching? *SoCG*, 2004.

[2] P. K. Agarwal, A. Efrat, and M. Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SICOMP*, 29(3):912–953, 2000.

[3] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Geometric approximation via coresets. *Combinatorial and Computational Geometry (MSRI publication)*, 52, 2005.

[4] G. Aggarwal, M. Datar, S. Rajagopalan, and M. Ruhl. On the streaming model augmented with a sorting primitive. In *FOCS*, 2004.

[5] A. Andoni, K. Do Ba, P. Indyk, and D. Woodruff. Efficient sketches for Earth-Mover Distance, with applications. In *FOCS*, 2009.

[6] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *JACM*, 45(5):753–782, 1998.

[7] B. Bahmani, R. Kumar, and S. Vassilvitskii. Densest subgraph in streaming and MapReduce. *VLDB*, 2012.

[8] P. Beame and J. Håstad. Optimal bounds for decision problems on the CRCW PRAM. *JACM*, 36(3):643–670, 1989.

[9] P. Beame, P. Koutris, and D. Suciu. Communication steps for parallel query processing. In *PODS*, 2013.

[10] G. E. Blelloch, R. Peng, and K. Tangwongsan. Linear-work greedy parallel approximate set cover and variants. 2011.

[11] P. B. Callahan and S. R. Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *SODA*, 1993.

[12] F. Chierichetti, R. Kumar, and A. Tomkins. Max-Cover in Map-Reduce. In *WWW*, 2010.

[13] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[14] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *CACM*, 51(1):107–113, 2008.

[15] A. Ene, S. Im, and B. Moseley. Fast clustering using MapReduce. In *KDD*, 2011.

[16] D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *STOC*, 2011.

[17] J. Feldman, S. Muthukrishnan, A. Sidiropoulos, C. Stein, and Z. Svitkina. On distributing symmetric streaming computations. *ACM Transactions on Algorithms*, 6(4), 2010. Previously in SODA'08.

[18] G. Frahling, P. Indyk, and C. Sohler. Sampling in dynamic data streams and applications. *International Journal of Computational Geometry & Applications*, 18(01n02):3–28, 2008. Previously in SoCG'05.

[19] M. T. Goodrich. Communication-efficient parallel sorting. *SICOMP*, 29(2):416–432, 1999. Previously in STOC'96.

[20] M. T. Goodrich, N. Sitchinava, and Q. Zhang. Sorting, searching, and simulation in the MapReduce framework. In *ISAAC*, 2011.

[21] K. Grauman and T. Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *ICCV*, Beijing, China, October 2005.

[22] S. Har-Peled, P. Indyk, and R. Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of Computing*, 8(1):321–350, 2012.

[23] P. Indyk. *High-dimensional Computational Geometry*. PhD thesis, Stanford University, 2000.

[24] P. Indyk. Algorithms for dynamic geometric problems over data streams. *STOC*, 2004.

[25] P. Indyk. A near linear time constant factor approximation for Euclidean bichromatic matching (cost). In *SODA*, 2007.

[26] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. *ACM SIGOPS Operating Systems Review*, 41(3):59–72, 2007.

[27] H. J. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *SODA*, 2010.

[28] J. Kleinberg and E. Tardos. *Algorithm design*. Pearson Education India, 2006.

[29] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani. Fast greedy algorithms in MapReduce and streaming. 2013.

[30] S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii. Filtering: a method for solving graph problems in MapReduce. 2011.

[31] Y. Rubner, C. Tomasi, and L. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

[32] R. Sharathkumar and P. K. Agarwal. Algorithms for the transportation problem in geometric settings. In *SODA*, 2012.

[33] R. Sharathkumar and P. K. Agarwal. A near-linear time approximation algorithm for geometric bipartite matching. In *STOC*, 2012.

[34] Y. Shiloach and U. Vishkin. An $O(\log n)$ parallel connectivity algorithm. *J. Algorithms*, 3(1):57–67, 1982.

[35] P. M. Vaidya. Minimum spanning trees in k-dimensional space. *SICOMP*, 17(3):572–582, 1988.

[36] P. M. Vaidya. Geometry helps in matching. *SICOMP*, 18(6):1201–1225, 1989.

[37] L. G. Valiant. A bridging model for parallel computation. *CACM*, 33(8):103–111, 1990.

[38] K. R. Varadarajan and P. K. Agarwal. Approximation algorithms for bipartite and non-bipartite matching in the plane. In *SODA*, 1999.

[39] T. White. *Hadoop: the definitive guide*. O'Reilly, 2012.

[40] C. T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Computers*, 100(1):68–86, 1971.