$O(1)$-Round MPC Algorithms for Multi-Dimensional Grid Graph Connectivity, Euclidean MST and DBSCAN

Junhao Gan
The University of Melbourne, Australia

Anthony Wirth
The University of Melbourne, Australia

Zhuo Zhang
The University of Melbourne, Australia

Abstract

In this paper, we investigate three fundamental problems in the *Massively Parallel Computation* (MPC) model: (i) *grid graph connectivity*, (ii) approximate *Euclidean Minimum Spanning Tree* (EMST), and (iii) approximate DBSCAN.

Our first result is a $O(1)$-round *Las Vegas* (i.e., succeeding with high probability) MPC algorithm for computing the *connected components* on a d-dimensional c-penetration grid graph ((d, c)-grid graph), where both d and c are positive integer constants. In such a grid graph, each vertex is a point with integer coordinates in $\mathbb{N}^d$, and an edge can only exist between two distinct vertices with ℓ_∞-norm at most c. To our knowledge, the current best existing result for computing the connected components (CC's) on (d, c)-grid graphs in the MPC model is to run the state-of-the-art MPC CC algorithms that are designed for general graphs: they achieve $O(\log\log n + \log D)$ [10] and $O(\log\log n + \log \frac{1}{\lambda})$ [8] rounds, respectively, where D is the *diameter* and λ is the *spectral gap* of the graph. With our grid graph connectivity technique, our second main result is a $O(1)$-round Las Vegas MPC algorithm for computing *approximate Euclidean MST*. The existing state-of-the-art result on this problem is the $O(1)$-round MPC algorithm proposed by Andoni et al. [5], which only guarantees an approximation on the overall weight *in expectation*. In contrast, our algorithm not only guarantees a *deterministic overall weight approximation*, but also achieves a *deterministic edge-wise weight approximation*. The latter property is crucial to many applications, such as finding the Bichromatic Closest Pair and Single-Linkage Clustering. Last, but not least, our third main result is a $O(1)$-round Las Vegas MPC algorithm for computing an *approximate DBSCAN* clustering in $O(1)$-dimensional Euclidean space.

2012 ACM Subject Classification Theory of computation → Massively parallel algorithms

Keywords and phrases Massively Parallel Computation, Graph Connectivity, Grid Graphs, Euclidean Minimum Spanning Tree, DBSCAN

Digital Object Identifier 10.4230/LIPIcs.ICDT.2025.7

Related Version *Full Version*: `https://arxiv.org/abs/2501.12044` [23]

Funding This work is in part supported by ARC DE190101118 and DP230102908.

1 Introduction

Effective parallel systems for large scale datasets, such as MapReduce [17], Dryad [27], Spark [37], Hadoop [34], have received considerable attention in recent years. The *Massively Parallel Computation* (MPC) model [28, 24, 9] has been proposed to provide a solid theoretical abstraction for the modern study of parallelism.

In this paper, we propose several $O(1)$-round Las Vegas algorithms, each succeeding with high probability, in the MPC model, for solving three fundamental problems: (i) *connectivity* and *minimum spanning forest* (MSF) on multi-dimensional grid graphs, (ii) *approximate Euclidean Minimum Spanning Tree* (EMST), and (iii) *approximate DBSCAN*.

The MPC Model. In the MPC model, the input is of size n and *evenly* distributed among m machines. Each machine is equipped with a *strictly sub-linear* local memory of size $\Theta(s)$, where $s = n/m = n^{\alpha}$ for some *constant* $\alpha \in (0, 1)$ [5, 28, 24]. An MPC algorithm runs in *(synchronous) rounds*; each round proceeds two phases *one after another* : the *communication phase* first, and then the *local computation phase*. In the communication phase, each machine can send to and receive from other machines, in total, $O(s)$ data. In the computation phase, each machine performs computation on the $O(s)$ data stored in its local memory, and prepares what data can be sent to which machine in the communication phase of the next round. The efficiency of an MPC algorithm is measured by the *total number of rounds*.

Main Result 1: Connectivity on Grid Graphs

We consider a class of graphs called *d-dimensional c-penetration grid graphs* (for short, (d, c)-grid graphs). Specifically, a graph $G = (V, E)$ is a (d, c)-grid graph if it satisfies: (i) the dimensionality $d \geq 2$ is a *constant* and $c \geq 1$ is an *integer*; (ii) each node in V is a d-dimensional point with *integer coordinates* in $\mathbb{N}^d$ space; (iii) for each edge $(u, v) \in E$, $0 < \|u, v\|_{\infty} \leq c$ holds, i.e., u and v are distinct vertices and their coordinates differ by at most c on every dimension. It can be verified that, when c is a constant, by definition, each node in a (d, c)-grid graph can have *at most* $(2c + 1)^d - 1 \in O(1)$ neighbours, and hence, a (d, c)-grid graph is *sparse*, i.e., $|E| \in O(|V|)$. In particular, for $c = 1$, a $(d, 1)$-grid graph is a common well-defined d-dimensional grid graph.

In real-world applications, grid graphs are often useful in the modeling of 3D meshes and terrains [11]. More importantly, grid graphs are extremely useful in many *algorithmic designs* for solving various computational geometry problems, such as Approximate Nearest Neighbour Search [7], Euclidean Minimum Spanning Tree [26], DBSCAN [4] and etc. Therefore, the study of algorithms on grid graphs has become an important topic in the research community. Our first result is this theorem:

▶ **Theorem 1 ([*]).** *Given a (d, c)-grid graph $G = (V, E)$ with $c \in O(1)$, there exists a Las Vegas MPC algorithm with local memory $\Theta(s) \subseteq \Theta(|V|^{\alpha})$ per machine, for an arbitrary constant $\alpha \in (\max\{\frac{d+1}{d+2}, \frac{4}{5}\}, 1)$, which computes all the* connected components *of G in $O(1)$ rounds* with high probability. *Specifically, the algorithm assigns the ID of the connected component to each node, of which the node belongs to.*

Computing connected components (CC's) of sparse graphs in the MPC model is notoriously challenging. In particular, the well accepted *2-CYCLE Conjecture* [36, 32] says that:

> Every MPC algorithm, using $n^{1-\Omega(1)}$ local memory per machine and $O(n)$ space in total, requires $\Omega(\log n)$ rounds to correctly solve, with high probability, the *1-vs-2-Cycle problem* which asks to distinguish whether the input graph consists of just one cycle with n nodes or two cycles with $n/2$ nodes each.

Theoretically speaking, the 2-CYCLE Conjecture can be disproved as "easy" as finding just a single constant $\alpha < 1$ and an algorithm that solves the 1-vs-2-Cycle problem with $O(n^{\alpha})$ per-machine local memory and $O(n)$ total space in $o(\log n)$ rounds. However, according to the recent hardness results, the conjecture is robust and hard to be refuted [32].

Even worse, since the input instance of the 1-vs-2-Cycle problem is so simple, it eliminates the hope of computing CCs for many classes of "simple" graphs, such as path graphs, tree graphs and planar graphs, which are often found to be simpler cases for many fundamental problems. As a result, it is still very challenging to find a large class of graphs on which the CC problem can be solved in $o(\log n)$ rounds.

State-of-the-art works [6, 8, 10, 13] consider general sparse graphs that are parameterised by the *diameter* D and the *spectral gap* λ. Specifically, Andoni et al. [6] propose a $O(\log D \log\log n)$-round randomized algorithm, where n is the number of nodes. Assadi et al. [8] give an algorithm with $O(\log\log n + \log\frac{1}{\lambda})$ rounds. Recently, Behnezhad et al. [10] improve the bound by Andoni et al. to $O(\log D + \log\log n)$ randomized, while Coy and Czumaj [13] propose a deterministic MPC algorithm achieving the same round number bound. However, all these state-of-the-art general algorithms still require $O(\log n)$ rounds for solving the CC problem on (d, c)-grid graphs in the worst case, as the diameter D of a (d, c)-grid graph can be as large as n and the spectral gap, λ, can be as small as $\frac{1}{n}$.

Therefore, our Theorem 1 is not only an immediate improvement over these state-of-the-art known results the CC problem on (d, c)-grid graphs, but, interestingly, also suggests a large class of sparse graphs, (d, c)-grid graphs with $c \in O(1)$, on which the CC problem can be solved in $O(1)$ rounds.

We note that it is the *geometric property* of (d, c)-grid graphs making them admit $O(1)$-round CC algorithms. As we shall discuss in Section 2, our key technique is the so-called *Orthogonal Vertex Separator* for multi-dimensional grid graphs proposed by Gan and Tao [22] which was originally proposed to efficiently compute CC's on $(d, 1)$-grid graphs in *External Memory* model [3]. However, in the MPC model, it appears difficult to compute such separators in $O(1)$ rounds. To overcome this, we relax the Orthogonal Vertex Separator to a "*weaker yet good-enough*" version for our purpose of designing $O(1)$-round MPC algorithms and extend this technique [1] to (d, c)-grid graphs which are crucial to our $O(1)$-round EMST and DBSCAN algorithms.

Moreover, with standard modifications, our CC algorithm can also compute the *Minimum Spanning Forests* (MSF) on (d, c)-grid graphs:

▶ **Corollary 2 ([*]).** *Given an* edge-weighted (d, c)*-grid graph* $G = (V, E)$ *with* $c \in O(1)$*, there exists a Las Vegas MPC algorithm with local memory* $\Theta(s) \subseteq \Theta(|V|^{\alpha})$ *per machine, for arbitrary constant* $\alpha \in (\max\{\frac{d+1}{d+2}, \frac{4}{5}\}, 1)$*, which computes a* Minimum Spanning Forest *in* $O(1)$ *rounds with high probability.*

Main Result 2: Approximate EMST

By enhancing our MSF technique for (d, c)-grid graphs, our second main result is a new $O(1)$-round Las Vegas MPC algorithm for computing approximate *Euclidean Minimum Spanning Tree* (EMST). Specifically, the problem is defined as follows. Given a set, P, of n points with integer coordinates in d-dimensional space $\mathbb{N}^d$, where d is a constant and the coordinate values are in $[0, \Delta]$ with $\Delta = n^{O(1)}$, let $G = (V, E)$ be an *implicit* edge-weighted complete graph on P such that: $V = P$ and the weight of each edge $(u, v) \in E$ is the Euclidean distance between u and v. The goal of the EMST problem is to compute an MST

[1] A similar technique of using small-size geometric separators (though they are not the same as ours) is applied to computing Contour Trees on terrains [33] in $O(1)$ rounds in the MPC model. Moreover, the idea of a crucial technique in our separator computation, ε-approximation, actually stems from the KD-tree construction in the MPC model in [2].

on the implicit complete graph, G, of P. The EMST problem is one of the most fundamental and important problems in computational geometry and has sparked significant attention for machine learning [18, 25, 30], data clustering [15, 14] and optimization problems [16, 31]. The existing state-of-the-art result is an $O(1)$-round *Monte-Carlo* MPC algorithm proposed by Andoni et al. [5] for computing an approximate EMST. Their algorithm possesses the following properties: (i) it can achieve $O(1)$ rounds in the worst case and works for all constant $\alpha \in (0, 1)$; and (ii) *in expectation*, it returns a spanning tree T of the implicit graph G whose total edge weight is at most $(1 + \rho)$ times of the total edge weight of the exact EMST T^*, where the approximation factor $\rho > 0$ is a *constant*. In comparison, we have the following theorem:

▶ **Theorem 3 ([*]).** *Given a set P of n points in d-dimensional space $\mathbb{N}^d$ with coordinate values in $[0, \Delta]$ for $\Delta = n^{O(1)}$ and a constant approximation factor $\rho > 0$, there exists a Las Vegas MPC algorithm with local memory $\Theta(s) \subseteq \Theta(n^\alpha)$ per machine, for arbitrary constant $\alpha \in (\max\{\frac{d+1}{d+2}, \frac{4}{5}\}, 1)$, which computes a spanning tree T of the implicit graph G of P in $O(1)$ rounds with high probability and T* deterministically *satisfyies the following:*

- **Overall Weight Approximation***: the total edge weight of T is at most $(1 + \rho)$ times the total edge weight of the exact EMST T^*;*
- **Edge-Wise Weight Approximation***: for every edge (u, v) in T^* with weight $w(u, v)$, there must exist a path from u to v in T such that the maximum weight of the edges on this path is at most $(1 + \rho) \cdot w(u, v)$.*

Our algorithm improves Andoni et al.'s approximate EMST algorithm in two folds. First, our algorithm computes a feasible ρ-approximate EMST deterministically, while theirs can only guarantee this in expectation. Second, and most importantly, the approximate EMST computed by our algorithm can further guarantee the edge-wise approximation which Andoni et al.'s algorithm cannot achieve. We note that the edge-wise approximation guarantee is crucial to solving many down-stream algorithmic problems, such as *ρ-approximate bichromatic closest pair* [1], Single-Linkage Clustering [38] and other related problems [35, 19].

Main Result 3: Approximate DBSCAN

Our third main result is a $O(1)$-round MPC algorithm for computing ρ-approximate DBSCAN clustering as defined in [20]. To our best knowledge, this is the first $O(1)$-round algorithm in the MPC model for solving the DBSCAN problem.

▶ **Theorem 4 ([*]).** *Given a set P of n points in d-dimensional space $\mathbb{R}^d$, where d is a constant, two DBSCAN parameters: ε and minPts, and a constant approximation factor $\rho > 0$, there exists a Las Vegas MPC algorithm with local memory $\Theta(s) \subseteq \Theta(n^\alpha)$ per machine, for arbitrary constant $\alpha \in (\max\{\frac{d+1}{d+2}, \frac{4}{5}\}, 1)$, which computes an ρ-approximate DBSCAN clustering of P in $O(1)$ rounds with high probability.*

Derandomization

As we shall see shortly, the randomness of all our Las Vegas algorithms only comes from the *randomized computation* of an *$\frac{1}{r}$-approximation* for some r. Indeed, it is known that such approximations can be computed in $O(1)$ MPC rounds deterministically [2]. And therefore, all our algorithms can be *derandomized*, yet slightly shrinking the feasible value range of the local memory parameter, α, to $(\max\{\frac{d+1}{d+2}, \frac{6}{7}\}, 1)$.

Full Paper

The proofs of all the theorems, lemmas and claims marked with [*] can be found in the full version of this paper, on arXiv [23].

2 Grid Graph Connectivity and MSF

In this section, we introduce our $O(1)$-round MPC algorithms for computing CC's and MSF's on (d, c)-grid graphs with $c \in O(1)$ with high probability.

Implicit Graphs and Implicit (d, c)-Grid Graphs. For the ease of explanation of our techniques for solving the EMST problem in the next section, we first introduce the notion of *implicit graphs*, denoted by $G = (V, \mathcal{E})$.

▶ **Definition 5** (Implicit Graph). *An* implicit graph $G = (V, \mathcal{E})$ *is a graph consisting of a* node set V *and an* edge formation rule, $\mathcal{E}$, *where:*

- *each node u in V is associated with $O(1)$ words of information, e.g., $O(1)$-dimensional coordinates;*
- *the edge formation rule, $\mathcal{E}$, is a function (of size bounded by $O(1)$ words) on distinct node pairs: for any two distinct nodes $u, v \in V$, $\mathcal{E}(u, v)$ returns, based on the $O(1)$-word information associated with u and v: (i) whether edge (u, v) exists in G, and (ii) the weight of (u, v) if edge (u, v) exists and the weight is well defined.*

By the above definition, the size of every implicit graph is bounded by $O(|V|)$. An implicit graph can be converted to an *explicit graph* by materialising all the edges with the edge formation rule $\mathcal{E}$. If the corresponding explicit graph of an implicit graph $G = (V, \mathcal{E})$ is a (d, c)-grid graph, we say $G = (V, \mathcal{E})$ is an *implicit (d, c)-grid graph*. For example, given a set P of n points in $\mathbb{N}^d$ with coordinate values in $[0, \Delta]$, the *implicit complete Euclidean graph* G_{comp} of P can be considered as an implicit (d, Δ)-grid graph $G = (P, \mathcal{E})$, where $\mathcal{E}(u, v)$ returns $dist(u, v)$, the Euclidean distance between u and v, as the edge weight. Another example is the unit-disk graph [12] in $\mathbb{N}^d$ space with l_∞-norm, which can be essentially regarded as the *implicit $(d, 1)$-grid graph* with an edge formation rule: there exists an edge between two vertices if and only if their l_∞-norm distance is at most 1. However, as we will see in our EMST algorithm, the notion of implicit (d, c)-grid graphs is more general due to the flexibility on the edge formation rule specification. For example, in addition to the coordinates, each node can also bring an ID of the connected component that it belongs to, and the edge formation rule can further impose a rule that there must be no edge between two nodes having the same connected component ID.

2.1 Pseudo s-Separator and Its Existence

In this subsection, we extend the notion of Orthogonal Vertex Separator [22] to a *weaker yet good-enough* version for our algorithm design purpose for solving the CC and MSF problems on implicit (d, c)-grid graphs in the MPC model. We call this new notion a *Pseudo s-Separator*; recall that $O(s)$ is the size of local memory, per machine, in the MPC model:

▶ **Definition 6** (Pseudo s-Separator). *Consider an implicit (d, c)-grid graph $G = (V, \mathcal{E})$ and a parameter $s < |V|$. A set $S \subseteq V$ is a* pseudo s-separator *of G if it satisfies:*

- $|S| \in O(\frac{c \cdot |V|}{s^{1/d}} \cdot \log s)$
- *Removing the vertices in S disconnects G into $h \in O(|V|/s)$ sub-graphs $G_i = (V_i, \mathcal{E})$, $i \in [1, h]$, such that $V_i \cap V_j = \emptyset$ for all $i \neq j$, $V_i \subset V$ and $|V_i| \in O(s)$.*

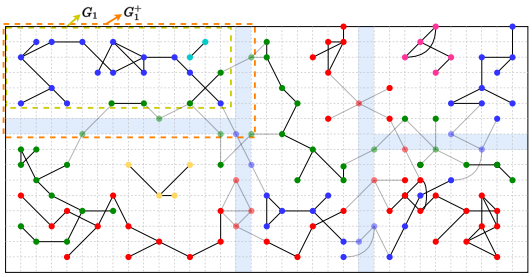

Figure 1 A (d, c)-grid graph G with $d = 2$ and $c = 2$ and a separator. The blue regions are the target c-dividers: the vertices in them are separator vertices. Removing all the separator vertices disconnects G into 5 sub-graphs.

Figure 1 shows an example. Next, we show that if $1 \leq c \leq s^{\frac{1}{d^3}}$ and $|V| > s$, a pseudo s-separator must exist in an implicit (d, c)-grid graph $G = (V, \mathcal{E})$. While our proof mimics that in [22], it does require some new non-trivial ideas. We introduce some useful notations.

Minimum Bounding Space of G. Consider an implicit (d, c)-grid graph $G = (V, \mathcal{E})$; let $[x_{\min}^{(i)}, x_{\max}^{(i)}]$ be the value range of V on the i-th dimension (for $i \in \{1, 2, \ldots, d\}$), where $x_{\min}^{(i)}$ and $x_{\max}^{(i)}$ are the *smallest* and *largest* coordinate values of the points in V on dimension i, respectively. The *minimum bounding space* of G is defined as $\text{mbr}(V) = [x_{\min}^{(1)}, x_{\max}^{(1)}] \times [x_{\min}^{(2)}, x_{\max}^{(2)}] \times \cdots \times [x_{\min}^{(d)}, x_{\max}^{(d)}]$. Moreover, we denote the projection of $\text{mbr}(V)$ on the i-th dimension by $\text{mbr}(V)[i] = [x_{\min}^{(i)}, x_{\max}^{(i)}]$.

***c*-Divider.** Consider the minimum bounding space $\text{mbr}(V)$; a *c-divider* on the i-th dimension at coordinate x, denoted by $\pi(i, x)$, is an axis-parallel rectangular range whose projection on dimension i is $[x, x + c - 1]$ and the projection on dimension $j \neq i$ is $\text{mbr}(V)[j]$. With a c-divider $\pi(i, x)$, the vertex set V is partitioned into three parts: (i) the *left part* $V_{\text{left}} = \{u \in V \mid u[i] \leq x - 1\}$, (ii) the *right part* $V_{\text{right}} = \{u \in V \mid u[i] \geq x + c\}$, and (iii) the *boundary part* $S_\pi = V \cap \pi(i, x)$, where $u[i]$ is the i^{th} coordinate of a vertex u.

We say that $\pi(i, x)$ *separates* G into two sub-graphs induced by the left and right part, i.e., V_{left} and V_{right}, respectively. And all the vertices in S_π are called *separator vertices*. Observe that for any $u \in V_{\text{left}}$ and any $v \in V_{\text{right}}$, $|u[i] - v[i]| \geq c + 1$ implying $\|u, v\|_\infty \geq c + 1$. As a result, there must not exist any edge between such u and v. With such notation, we show the following Binary Partition Lemma.

▶ **Lemma 7** (Binary Partition Lemma). *Consider an implicit (d, c)-grid graph $G = (V, \mathcal{E})$ with $1 \leq c \leq s^{\frac{1}{d^3}}$ and $|V| > s$; there exists a c-divider $\pi(i, x)$ partitioning V into $\{S_\pi, \{V_{left}, V_{right}\}\}$ such that:*

- $|V_{left}| \geq \frac{|V|}{4(d+1)}$ *and* $|V_{right}| \geq \frac{|V|}{4(d+1)}$, *and*
- $|S_\pi| \leq 2c(1 + d)^{\frac{1}{d}} |V|^{1 - \frac{1}{d}}$.

Proof. We prove this lemma with a constructive algorithm. Given the vertex set, V, our algorithm first finds two integers, y_j and z_j, for each dimension $j \in [1, d]$. Specifically, y_j is the *largest* integer that the number of the vertices in V located to the "left" of y_j is at most $\frac{|V|}{2(1+d)}$, while z_j is the *smallest* integer that the number of the vertices in V located to the "right" of z_j is at most $\frac{|V|}{2(1+d)}$. Formally, $y_j = \max\{x \in \mathbb{N} : |\{v \in V \mid v[j] < x\}| \leq \frac{|V|}{2(1+d)}\}$ and $z_j = \min\{x \in \mathbb{N} : |\{v \in V \mid v[j] > x\}| \leq \frac{|V|}{2(1+d)}\}$, where $v[j]$ is the coordinate of v on dimension j.

Consider the hyper-box in $\mathbb{N}^d$, whose projection on each dimension is $[y_j, z_j]$ for $j \in [1, d]$. By the definition of y_j and z_j, this box contains in total at least $|V|(1-\frac{2d}{2(d+1)}) = \frac{|V|}{d+1}$ vertices in V. Since the coordinates of the points are all integers, the volume of the box must be at least $\frac{|V|}{d+1}$. That is, $\prod_{j=1}^{d}(z_j - y_j + 1) \geq \frac{|V|}{d+1}$. Thus, there must exist some dimension i such that $z_i - y_i + 1 \geq (\frac{|V|}{1+d})^{\frac{1}{d}}$. Next we fix this dimension, i. Let $S_{\pi(i,x)}$ be the set of all vertices in V falling in a c-divider $\pi(i, x)$. Within dimension i, each vertex in V can be contained in at most c *consecutive* c-dividers. As a result, we have $\sum_{x' \in [y_i, z_i - c]} |S_{\pi(i,x')}| \leq c|V|$. Since there are $z_i - c - y_i + 1$ terms in the summation, there exists an integer $x \in [y_i, z_i - c]$ with

$$|S_{\pi(i,x)}| \leq \frac{c|V|}{z_i - y_i + 1 - c} \leq \frac{c|V|}{(\frac{|V|}{1+d})^{\frac{1}{d}} - c} \leq 2c(1+d)^{\frac{1}{d}}|V|^{1-\frac{1}{d}},$$

where the second inequality comes from the bound on $z_i - y_i + 1$ and the third inequality is due to $c \leq s^{\frac{1}{d^3}} \leq \frac{1}{2} \cdot (\frac{|V|}{(1+d)})^{\frac{1}{d}}$ as $|V| > s$. Then such $\pi(i, x)$ is a desired c-divider.

Next, we bound the size of V_{left} with respect to $\pi(i, x)$: bounding $|V_{\text{right}}|$ is analogous. Since $x \geq y_i$, there are only two possible relations between x and y_i:

- If $x > y_i$, according to the definition of y_i, the number of vertices in V to the left of π is at least $|V|(\frac{1}{2d+2}) \geq \frac{|V|}{4d+4}$.
- If $x = y_i$, the number of vertices in V to the left of π is at least $\frac{|V|}{2d+2} - |S_{\pi(i,x)}| \geq |V|\left(\frac{1}{2d+2} - \frac{2c(d+1)^{\frac{1}{d}}}{|V|^{\frac{1}{d}}}\right) \geq \frac{|V|}{4d+4}$.

Either way, $|V_{\text{left}}| \geq \frac{|V|}{4(d+1)}$ holds; Lemma 7 thus follows. ◀

The Multi-Partitioning Algorithm. Given an implicit (d, c)-grid graph $G = (V, \mathcal{E})$ with $1 \leq c \leq s^{\frac{1}{d^3}}$ and $|V| > s$, initialize $S \leftarrow \emptyset$ and perform the following steps:

- apply Lemma 7 on G to obtain $\{S_\pi, \{V_{\text{left}}, V_{\text{right}}\}\}$;
- $S \leftarrow S \cup S_\pi$;
- if $|V_{\text{left}}| > s$, recursively apply Lemma 7 on $G_{\text{left}} = (V_{\text{left}}, \mathcal{E})$;
- if $|V_{\text{right}}| > s$, recursively apply Lemma 7 on $G_{\text{right}} = (V_{\text{right}}, \mathcal{E})$;

▶ **Lemma 8 ([*]).** *The vertex set S, obtained by the above* Multi-Partitioning Algorithm, *is a pseudo s-separator of the implicit (d, c)-grid graph $G = (V, \mathcal{E})$, where $1 \leq c \leq s^{\frac{1}{d^3}}$ and $|V| > s$.*

A construction algorithm proves existence, hence:

▶ **Theorem 9.** *For any implicit (d, c)-grid graph $G = (V, \mathcal{E})$ with $1 \leq c \leq s^{\frac{1}{d^3}}$ and $|V| > s$, a pseudo s-separator S of G must exist.*

Remark. While our Pseudo s-Separator (PsSep) looks similar to the notation of the Orthogonal Vertex Separator (OVSep) proposed by Gan and Tao [22], we emphasize that some crucial differences between them are worth noticing. First, the OVSep was originally proposed for solving problems on $(d, 1)$-grid graphs (in the External Memory model). Our PsSep supports implicit (d, c)-grid graphs for parameter c up to a non-constant $s^{\frac{1}{d^3}}$ in the MPC model. This extension, discussed in Section 3, plays a significant role in our $O(1)$-round algorithm for computing Approximate EMST's. Second and most importantly, it is still unclear how (and appears difficult) to compute the OVSep in $O(1)$ MPC rounds. In contrast, our PsSep is proposed to overcome this technical challenge and can be computed in $O(1)$ MPC rounds. However, as discussed in the next sub-section, because of the application of ε-approximation in the construction algorithm of PsSep, the separator size at each recursion

level no longer geometrically decreases and therefore, leading to a logarithmic blow-up in the overall separator size in PsSep. Despite of this size blow-up, by setting the parameters carefully, we show that our PsSep can still fit in the local memory of one machine, and thus, it is still *good enough* for the purpose of computing CC's on implicit (d, c)-grid graphs and solving the Approximate EMST and Approximate DBSCAN problems in $O(1)$ rounds in the MPC model.

2.2 $O(1)$-Round Pseudo s-Separator Algorithm

In this subsection, we show a $O(1)$-round MPC algorithm for computing a pseudo s-separator for implicit (d, c)-grid graphs with $1 \leq c \leq s^{\frac{1}{d^3}}$. The Multi-Partitioning Algorithm proves the existence of a pseudo s-separator, but a straightforward simulation of this algorithm in the MPC model is, however, insufficient to achieve $O(1)$ rounds. To overcome this, we resort to the technique of *ε-approximation* [29, 2].

2.2.1 Preliminaries

Range Space and ε-Approximation. A *range space* Σ is a pair $(X, \mathcal{R})$, where X is a ground set and $\mathcal{R}$ is a family of subsets of X. The elements of X are *points* and the elements of $\mathcal{R}$ are *ranges*. For $Y \subseteq X$, the *projection* of $\mathcal{R}$ on Y is $\mathcal{R}_{|Y} = \{q \cap Y \mid q \in \mathcal{R}\}$. Given a range space $\Sigma = (X, \mathcal{R})$ and $\varepsilon \in [0, 1]$, a subset $X' \subseteq X$ is called an *ε-approximation* of Σ if for every range $q \in \mathcal{R}$, we have $\left| \frac{|X' \cap q|}{|X'|} - \frac{|q|}{|X|} \right| \leq \varepsilon$.

▶ **Fact 1** ([29, 2]). *Consider the range space $(V, \mathcal{R})$, where the ground set V is a set of n points in $\mathbb{N}^d$ and $\mathcal{R} = \{V \cap \square \mid \forall$ axis-parallel rectangular range $\square$ in $\mathbb{N}^d\}$. A random sample set $\tilde{V}$ of V with size $|\tilde{V}| \in \Theta(r^2 \cdot \log n)$ is a $\frac{1}{r}$-approximate of $(V, \mathcal{R})$ with high probability.*

Throughout this paper, we consider axis-parallel rectangular ranges only. For simplicity, we may just say $\tilde{V}$ is a $\frac{1}{r}$-approximation of V.

2.2.2 Our Pseudo s-Separator Algorithm

Our MPC algorithm comprises several *super rounds*, each performing:

- Obtain an ε-approximation, $\tilde{V}$, tuning ε so this fits in the local memory of a machine;
- Perform roughly $O(s^{O(1)})$ "good enough" binary partitions in the local memory.

Our algorithm then *simultaneously* recurs on each of the resulting induced sub-graphs, if applicable, in the next super round. As we will show, each super round would decrease the size of the graph by a factor of $\Theta(s^{O(1)})$, and there can be at most $O(\log_{s^{O(1)}} \frac{|V|}{s}) = O(1)$ super rounds. Moreover, by Fact 1, an ϵ-approximation can be obtained by sampling which can be achieved in $O(1)$ MPC rounds; detailed implementations can be found in the appendix of our full paper [23]. Therefore, our algorithm runs in $O(1)$ MPC rounds.

Define $r = 2s^{1/d}$. In order to ensure that a $\frac{1}{r}$-approximation $\tilde{V}$ of V fits the local memory size, $O(s)$, without loss of generality, in the following, we assume that the dimensionality, d, is at least 3. Otherwise, for the case $d = 2$, we can "lift" the dimensionality to 3 by adding a *dummy* dimension to all the points in V; however, the bounds derived for $d = 3$ apply.

The challenge lies in how to find a *good enough* (compared to that in Lemma 7) c-divider π to separate V in the local memory with access to a $\frac{1}{r}$-approximation $\tilde{V}$ *only*.

▶ **Lemma 10 ([*]).** *Given V with $|V| > s$ and $d \geq 3$, with only access to a $\frac{1}{r}$-approximation $\tilde{V}$ of V such that $r = 2s^{1/d}$ and $|\tilde{V}| \in O(s)$, we can find a c-divider π in $mbr(V)$, which satisfies:*

- $|V_{left}| \geq \frac{|V|}{8(d+1)}$ *and* $|V_{right}| \geq \frac{|V|}{8(d+1)}$,
- $|S_\pi| \leq 4c(1+d)^{1/d}|V|s^{-1/d}$.

Lemma 10 suggests that, with only access to a $\frac{1}{r}$-approximation of V, we can compute a *good enough* c-divider π; it separates V into two parts each with size $\geq \frac{|V|}{8(d+1)}$; the size of the separator S_π is still bounded by $O(\frac{c \cdot |V|}{s^{1/d}})$, the same as in Lemma 7.

By a more careful analysis, interestingly, with a sample set $\tilde{V}$ of size $\Theta(s)$, one can derive that $\tilde{V}$ indeed is sufficient to be, with high probability, a $\frac{1}{3r^{1+l}}$-approximation, with some constant $l \in (0, \frac{1}{2})$. The $\frac{1}{3r^{1+l}}$-approximation property of $\tilde{V}$ is sufficient to invoke Lemma 10 for multiple times, i.e., making multiple partitions, with $\tilde{V}$ in local memory. We formalise this with the following crucial lemma.

▶ **Lemma 11 ([*]).** *Consider a $\frac{1}{3r^{1+l}}$-approximation $\tilde{V}$ of the range space $\Sigma = (V, \mathcal{R})$, where $|V| > r^l s$ and $l = \min\{\frac{1}{3}, \log_r \frac{|V|}{s}\}$ and $\mathcal{R}$ is the set of all possible axis-parallel rectangular ranges in $mbr(V)$. Let $\square$ be an arbitrary range in $\mathcal{R}$. Define $V_1 = \square \cap V$ and $\tilde{V}_1 = \square \cap \tilde{V}$.*

If $|\tilde{V}_1| \geq \frac{2|\tilde{V}|}{r^l}$, then we have: (i) $\tilde{V}_1$ is a $\frac{1}{r}$-approximation of $\Sigma' = (V_1, \mathcal{R}_1)$, where $\mathcal{R}_1 = \{q \in \mathcal{R} \mid q \subseteq V_1\}$, and (ii) $|V_1| > s$. Therefore, Lemma 10 is applicable on V_1 and $\tilde{V}_1$.

Our MPC Algorithm for Computing Pseudo s-Separator

One Super Round Implementation. Based on Lemma 11, given an implicit (d, c)-grid graph $G = (V, \mathcal{E})$, where $d \geq 3$, $|V| > r^l s$ for $l = \min\{\frac{1}{3}, \log_r \frac{|V|}{s}\}$, $1 \leq c \leq s^{\frac{1}{d^3}}$ and $r = 2s^{1/d}$, the implementation of one super round is shown in Algorithm 1.

Algorithm 1 One Super Round Implementation for Pseudo s-Separator.

Input: an implicit (d, c)-grid graph $G = (V, \mathcal{E})$ stored in a set of contiguous machines
Compute a random sample set $\tilde{V}$ of V in size $\Theta(s)$, send it to machine M_0;
M_0 processes locally:
  Let $K \leftarrow \frac{2|\tilde{V}|}{r^l}$, $\mathbb{V} \leftarrow \{\tilde{V}\}$, $\Pi \leftarrow \emptyset$;
  while $\exists \tilde{V}_1 \in \mathbb{V}$ *s.t.* $|\tilde{V}_1| \geq K$ **do**
    Apply Lemma 10 on $\tilde{V}_1$; denote the resulting c-divider by $\tilde{\pi}$, which partitions $\tilde{V}_1$ into $\{\tilde{S}_{\tilde{\pi}}, \{\tilde{V}_{\text{left}}, \tilde{V}_{\text{right}}\}\}$;
    Let $\mathbb{V} \leftarrow \mathbb{V} \cup \{\tilde{V}_{\text{left}}, \tilde{V}_{\text{right}}\} - \{\tilde{V}_1\}$, $\Pi \leftarrow \Pi \cup \{\tilde{\pi}\}$;
Broadcast Π to all machines related to V;
Each machine processes locally:
  Identify the separator vertices stored in their local memory with Π;
  Group vertices according to the induced sub-graph, according to Π;
After sorting, each sub-graph induced by Π is stored in a set of contiguous machines;

The Overall Pseudo s-Separator Algorithm. Given an implicit (d, c)-grid graph $G = (V, \mathcal{E})$, where $d \geq 3$, $|V| > r^l s$ for $l = \min\{\frac{1}{3}, \log_r \frac{|V|}{s}\}$, $1 \leq c \leq s^{\frac{1}{d^3}}$ and $r = 2s^{1/d}$,

- perform a new super round *simultaneously* on each of the induced sub-graphs G' with $|V'| > r^{l'} s$ for $l' = \min\{\frac{1}{3}, \log_r \frac{|V'|}{s}\}$;
- terminate when no more super rounds can be performed, in which case, all the induced sub-graphs have no more than s vertices.

▶ **Lemma 12 ([*]).** *After a super round, the input implicit (d, c)-grid graph $G = (V, \mathcal{E})$ is separated into h disconnected induced sub-graphs $G_i = (V_i, \mathcal{E})$ for $i \in [1, h]$ such that $h \in O(r^l)$ and $|V_i| \in O(|V|/r^l)$ for all $i \in [1, h]$.*

Algorithm 2 Connected Component Algorithm.

Input: implicit (d, c)-grid graph $G = (V, \mathcal{E})$ with $c \in [1, s^{\frac{1}{d^3}}]$
Output: for each vertex $v \in V$, the id of the connected component where v belongs

compute the pseudo s-separator with the algorithm in Section 2.2;
process each extended sub graph G_i^+ in their machine:
  compute the MSF of G_i^+, denote the edge set as T_i;
  compress T_i into an edge set E_i', s.t. it induces a tree. All leaf nodes of the tree are in S_i, and the internal nodes of the tree are either from S or have degree > 2;
  send E_i' to machine M_0;
M_0 run:
  build a graph based on all of the received edges $\cup E_i'$;
  solve the CC problem on E_i', get the CC id of vertices in S;
  send S_i back to the corresponding machine along with their CC id's;

▶ **Lemma 13 ([*]).** *When our Overall Pseudo s-Separator algorithm terminates, let S be the set of all the vertices in V falling in target c-dividers in applications of Lemma 10. Set S is a pseudo s-separator of the input implicit (d, c)-grid graph G.*

▶ **Lemma 14 ([*]).** *The total number of super rounds of our overall pseudo s-separator algorithm is bounded by $O(\frac{d}{\alpha})$, and the total number of MPC rounds is bounded by $O(1)$.*

Putting the above lemmas together, we have:

▶ **Theorem 15.** *Given an implicit (d, c)-grid graph $G = (V, \mathcal{E})$ with $d \geq 3$ and $1 \leq c \leq s^{\frac{1}{d^3}}$, there exists a Las Vegas MPC algorithm with local memory $\Theta(s) \subseteq \Theta(|V|^\alpha)$ per machine, for all constant $\alpha \in (0, 1)$, which computes a pseudo s-separator of G in $O(1)$ rounds with high probability.*

2.3 Grid Graph Connectivity Algorithms

Next, we introduce our MPC algorithms for computing CCs and MSFs for implicit (d, c)-grid graphs with $1 \leq c \leq s^{\frac{1}{d^3}}$.

Bounding the Feasible Range for α. Recall that Theorem 15 shows that a pseudo s-separator S of G can be computed in $O(1)$ rounds with high probability. This algorithm works for all constant $\alpha \in (0, 1)$. However, as we shall see shortly, both our CC and MSF algorithms require the pseudo s-separator, S, to fit in the local memory $O(s)$ in a single machine. As a result, they require $|S| \leq s$. From the proofs of Lemma 10 and Lemma 13, we know that $|S| \leq \frac{4c|V|}{s^{1/d}} \log s$. Recalling that $s = |V|^\alpha$, our algorithm works when $\alpha \geq \frac{d+1}{d+2}$ for $d \geq 3$. Combining the case of $d = 2$, a feasible range of α becomes $(\max\{\frac{d+1}{d+2}, \frac{4}{5}\}, 1)$.

Connectivity and Minimum Spanning Forest. Consider a pseudo s-separator S of G; let $G_i = (V_i, \mathcal{E})$ be the induced sub-graphs of G by S, where $i \in [1, h]$. By Definition 6, we have $h = O(\frac{|V|}{s})$ and $|V_i| \in O(s)$. For each G_i, denote the projection of $\text{mbr}(V_i)$ on dimension j by

$[x_i^{(j)}, y_i^{(j)}]$, for $i \in [1, h]$ and $j \in [1, d]$. Consider the hyper-box H_i whose projection on each dimension j is $[x_i^{(j)} - c, y_i^{(j)} + c]$ for $i \in [1, h]$ and $j \in [1, d]$; we say that H_i is the *extended region* of G_i. Moreover, we define the *extended graph* of G_i, denoted by $G_i^+ = (V_i^+, \mathcal{E})$, as the induced graph by the vertices in $V_i^+ = V_i \cup S_i$, where $S_i = S \cap H_i$ is the set of separator vertices falling in the extended region of G_i. Figure 1 shows an example of G_i and G_i^+. Clearly, by choosing a constant $\alpha \in (\max\{\frac{d+1}{d+2}, \frac{4}{5}\}, 1)$, we know that $|S| \in O(s)$ and hence, $|V_i^+| \in O(s)$. Therefore, G_i^+ fits in the local memory in one machine. Moreover, without loss of generality, we assume that G_i^+ is stored completely in a machine M_i.

By incorporating the techniques of the existing External Memory CC and MSF algorithms for grid graphs [22], where the detailed MPC implementations of them are respectively shown in Algorithm 2 and Algorithm 3 in Appendix A, we have the following theorem:

▶ **Theorem 16 ([*]).** *Given an implicit (edge-weighted) (d, c)-grid graph $G = (V, \mathcal{E})$ with $1 \leq c \leq s^{\frac{1}{d^3}}$, there exists a Las Vegas MPC algorithm with local memory $\Theta(s) \subseteq \Theta(|V|^\alpha)$ per machine, for arbitrary constant $\alpha \in (\max\{\frac{d+1}{d+2}, \frac{4}{5}\}, 1)$, which computes all the connected components (resp., minimum spanning forest) of G in $O(1)$ rounds with high probability.*

3 $O(1)$-Round Approximate EMST

An Overview of Our ρ-Approximate EMST Algorithm. The basic idea of our ρ-approximate EMST algorithm is to mimic the Kruskal's algorithm. Specifically, our algorithm firstly uses those "short" edges to connect different connected components that are found so far, and then gradually considers "longer" edges. Our algorithm runs in super rounds, each of which takes $O(1)$ MPC rounds. In the i-th super round, our algorithm constructs a representative implicit (d, c)-grid graph $G_i = (V_i, \mathcal{E}_i)$ which only considers those edges with weight $\leq l_i = c^i(\frac{\rho}{\sqrt{d}})^{i-1}$, where $c = s^{\frac{1}{d^3}}$. Next, it invokes our $O(1)$-round MSF algorithm on G_i, and then starts a new super round repeating this process until a spanning tree T is obtained.

The Algorithm Steps. Consider a set P of n points in d-dimensional space $\mathbb{N}^d$ with coordinate value range $[0, \Delta]$. Let $V_1 = P$ and set $c = s^{\frac{1}{d^3}}$. Our MPC ρ-approximate EMST algorithm constructs a series of implicit (d, c)-grid graphs, denoted by $G_i = (V_i, \mathcal{E}_i)$, for $i \in \{1, 2, \ldots\}$.

- **Augmented Node Information:** Each vertex u in V_i is associated with two pieces of $O(1)$-size information: (i) a connected component id of u, denoted by $u.id$, and (ii) the original point $p \in P$ which u currently represents, denoted by $u.pt$.
- **Edge Formation Rule:** for any two distinct nodes $u, v \in V_i$, in building, $\mathcal{E}_i$: the edge (u, v) exists *if and only if* the Euclidean distance $dist(u, v) \leq l_i$, where $l_i = c^i(\frac{\rho}{\sqrt{d}})^{i-1}$; if it exists, its weight, $w(u, v)$, is 0 if $u.id = v.id$, otherwise $dist(u, v)$.

Initially, $V_1 \leftarrow P$ and for each node $u \in V_1$, $u.id$ is the id of the point and $u.pt$ is the point itself. Our ρ-approximate EMST algorithm runs in super rounds; in the i-th super round (for $i = 1, 2, \ldots$), it works as follows:

- **Solve Stage: MSF Computation on $G_i = (V_i, \mathcal{E}_i)$:**
  - invoke our $O(1)$-round MPC algorithm (in Theorem 16) to compute an MSF of G_i. In particular, let $u.cid$ be a temporary variable to denote the *current* connected component id of $u \in V_i$ in G_i. Observe the difference between $u.id$ and $u.cid$: the former records the connected component id of u in the *previous* super round, and is used to determine edge weights in $\mathcal{E}_i$ in the current round;

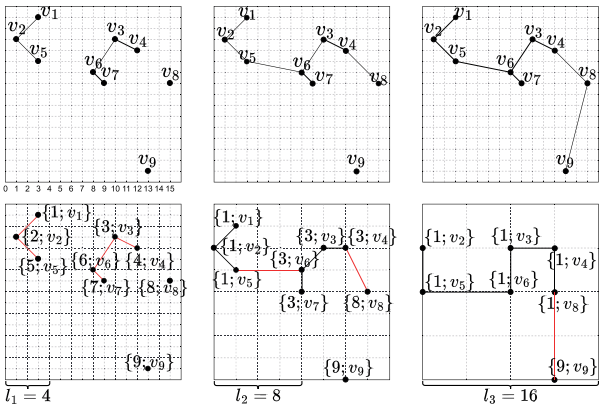

Figure 2 A running example of our approximate EMST algorithm.

- let $\mathcal{T}_i'$ be the set of edges in the MSF of G_i; remove edges with *zero* weight from $\mathcal{T}_i'$;
- initialize $\mathcal{T}_i \leftarrow \varnothing$; for each edge $(x, y) \in \mathcal{T}_i'$, add an edge $(x.pt, y.pt)$ to $\mathcal{T}_i$;
- keep tract of the total edge number that are added to $T = \cup_i \mathcal{T}_i$; when, in total, $n - 1$ edges are added, stop and return T as an ρ-approximate EMST of P;

- **Sketch Stage: Computing V_{i+1} from V_i:**
  - initialize $V_{i+1} \leftarrow \varnothing$; impose a grid on V_i with side length of $\frac{l_i \rho}{\sqrt{d}}$;
  - for each non-empty grid cell, g, add the most *bottom-left* corner point t of g to V_{i+1}; moreover, set $t.pt \leftarrow u.pt$ and $t.id \leftarrow u.cid$, where u is an arbitrary node in $V_i \cap g$;

MPC Implementation Details.

- At the beginning of the Solve Stage, the vertex set V_i is distributed across the machines. Thus, the MSF algorithm suggested in Theorem 16 is invoked on $G_i = (V_i, \mathcal{E}_i)$, and the resulted set of edges, $\mathcal{T}_i'$, is distributedly stored across the machines. By local computations, the edges in $\mathcal{T}_i'$ can be converted to edges in $\mathcal{T}_i$.
- To implement the Sketch Stage (i.e., computing V_{i+1} from V_i), each machine first computes the grid cell coordinate for each vertex in V_i locally. Our algorithm sorts all the vertices in V_i by their grid cell coordinates. After sorting, all the vertices falling in the same grid cell g are either stored in just one machine or a contiguous sequence of machines. Next, each machine generates the most bottom-left corner point t, according to our algorithm, for each grid cell g in its local memory. Duplicate points t are then removed by Duplicate Removal, an atomic MPC operation whose implementation details can be found in Appendix A. The vertex set V_{i+1} for the next super round is thus constructed.

A Running Example. Figure 2 gives a running example of our approximate EMST algorithm. The input set $P = \{v_1, \ldots, v_9\}$ contains nine points in 2-dimensional space with integer coordinate values in $[0, 15]$. Our algorithm solves this problem in three super rounds. The top row of the figures shows the edges that are labeled to be in the result edge set $\cup_i \mathcal{T}_i$ at the end of the round i. The bottom row shows the vertex set of V_i and the MSF of G_i.

In this example, $l_1 = 4$ and $\rho = \frac{\sqrt{2}}{2}$. In the first round, our algorithm constructs $G_1 = (V_1, \mathcal{E}_1)$. The vertex set V_1 with the augmented node information is shown in the first figure in at the bottom. The MSF of G_1, denoted as $\mathcal{T}_1'$, is computed and its edges are shown in the figure. After the computation of the MSF, each vertex $u \in V_1$ computes its

component id $u.cid$. All the zero-weight edges are deleted from $\mathcal{T}_1'$ and the remaining edges are highlighted in colour red. For each edge $(u_1, u_2) \in \mathcal{T}_1'$, its corresponding edge $(u_1.pt, u_2.pt)$ is marked as an output edge, i.e., the edge in the approximate EMST T returned by our algorithm, as shown in the first top-row figure. Next, the sketch stage starts; $G_2 = (V_2, \mathcal{E}_2)$ is thus constructed as follows. A grid with side length of $\frac{l_1\rho}{\sqrt{d}} = 2$ is imposed on V_1, and then V_2 is generated, shown in the second bottom-row figure. At the end of this second round, two new edges are added to T, the final output edge set of the approximate EMST. The same process is repeated with $l_2 = 8$ and $l_3 = 16$ in the third round, and one more edge is added to T. Since T now contains in total eight edges, it is returned as an approximate EMST on the input nine points.

Remark on $G_i = (V_i, \mathcal{E}_i)$. From the construction of G_i in our algorithm, at first glance, G_i is an implicit $(d, \frac{c \cdot l_i \rho}{\sqrt{d}})$-grid graph which looks ineligible for our MSF algorithm proposed in Theorem 16. However, observe that since the coordinate values of the nodes in V_i are all multiples of $\frac{l_i\rho}{\sqrt{d}}$, therefore, by a simple scaling, G_i is indeed an implicit (d, c)-grid graph.

▶ **Lemma 17 ([*]).** *Our ρ-approximate EMST algorithm takes $O(1)$ MPC rounds with high probability.*

Let T be the union of $\mathcal{T}_i$ for all $i = 1, 2, \ldots$. We prove that T is an (2ρ)-approximate EMST of P, achieving both overall and edge-wise approximation. By decreasing the constant approximation parameter ρ by half, an ρ-approximate EMST can be obtained.

▶ **Lemma 18 ([*]).** *T is an Euclidean spanning tree of P.*

▶ **Lemma 19** (Edge-Wise Approximation Guarantee **[*]**). *Consider an exact EMST T^* of P; for any edge $(u, v) \in T^*$, there must exist a path $\mathcal{P}(u, v)$ from u to v in T such that every edge (x, y) on this path has weight $w(x, y) \leq (1 + 2\rho) \cdot w(u, v)$.*

▶ **Lemma 20** (Overall Approximation Guarantee **[*]**). *Consider an exact EMST T^* of P; the total edge weight of T is at most $(1 + 2\rho)$ times the total edge weight of T^*.*

4 $O(1)$-Round Approximate DBSCAN

4.1 DBSCAN and ρ-Approximate DBSCAN

Core and Non-Core. Let P be a set of n points in d-dimensional Euclidean space $\mathbb{R}^d$, where the dimensionality $d \geq 2$ is a constant. For any two points $u, v \in P$, the Euclidean distance between u and v is denoted by $\text{dist}(u, v)$. There are two parameters: (i) a radius $\varepsilon \geq 0$ and (ii) core point threshold $minPts \geq 1$ which is a constant integer. For a point $v \in P$, define $B(v, \varepsilon)$ as the ball center at v with radius ε. If $B(v, \varepsilon)$ covers at least $minPts$ points in P, then v is a *core* point. Otherwise, v is a *non-core* point.

Primitive Clusters and Clusters. Consider an implicit *core graph* $G = (P_{\text{core}}, \mathcal{E})$, where P_{core} is the set of all the core points in P, and there exists an edge between two core points u and v if $\text{dist}(u, v) \leq \varepsilon$. Each connected component C of G is defined as a *primitive cluster*. For a primitive cluster C, let C' be the set of all the non-core points $p \in P$ such that $\text{dist}(p, C) = \min_{u \in C} \text{dist}(p, u) \leq \varepsilon$. The union of $C \cup C'$ is defined as *DBSCAN cluster*.

The DBSCAN Problem. Given a set of points P, a radius $\varepsilon \geq 0$ and a core threshold $minPts \geq 1$, the goal of the DBSCAN problem is to compute all the DBSCAN clusters.

ρ-Approximate DBSCAN. Given an extra approximation parameter, $\rho \geq 0$, the only difference is in the definition of the primitive clusters. Specifically, consider a conceptual *ρ-approximate core graph* $G_\rho = (P_{\text{core}}, \mathcal{E}_\rho)$, where the edge formation rule $\mathcal{E}_\rho$ works as follows. For any two core points $u, v \in P_{\text{core}}$: if $\text{dist}(u, v) \leq \varepsilon$, $\mathcal{E}_\rho(u, v)$ must return that (u, v) is in G_ρ; if $\text{dist}(u, v) > (1 + \rho)\varepsilon$, $\mathcal{E}_\rho(u, v)$ must return that (u, v) does not exist in G_ρ; otherwise, what $\mathcal{E}_\rho(u, v)$ returns does not matter.

Due to the does-not-matter case, G_ρ is not unique. Consider a graph G_ρ; each connected component C of G_ρ is defined as a primitive cluster with respect to G_ρ. And an ρ-approximate DBSCAN cluster is obtained by including non-core points to a primitive cluster C in the same way as in the exact version. The goal of the ρ-approximate DBSCAN problem is to compute the ρ-approximate DBSCAN clusters with respect to an arbitrary G_ρ.

4.2 Our MPC Algorithm

Our approximate DBSCAN algorithm consists of three main steps: (i) identify core points; (ii) compute primitive clusters from a G_ρ; and (iii) assign non-core points to primitive clusters. As shown in the existing ρ-approximate DBSCAN algorithm in [21], the third step, assigning non-core points, can be achieved by an analogous algorithm of core point identification. Next, we give an MPC algorithm for the first two steps.

4.2.1 Identify Core Points

We first show how to identify all the core points for a DBSCAN problem instance in $O(\frac{1}{\alpha})$ MPC rounds. Our goal is to label each point $p \in P$, as to whether or not it is a *core point*. In the following, we assume that the constant parameter α in the MPC model satisfies $\alpha \in (\frac{1}{2}, 1)$, and hence, $m \leq s$, that is, the number of machines is no more than the local memory size s.

Consider a grid imposed in the data space $\mathbb{R}^d$ with grid-cell length $l = \varepsilon/\sqrt{d}$. Each grid cell, g, is represented by the coordinates $(x_1, x_2, \ldots, x_d)$ of its *left-most corner*, namely, each grid cell g is essentially a hyper cube $[x_1 l, (x_1 + 1)l) \times \cdots \times [x_d l, (x_d + 1)l)$. Each point $p \in P$ is contained in one and exactly one grid cell, denoted by $g(p)$. The size of a grid cell g, denoted by $|g|$, is defined as the number of points in P contained in g. If $|g| > 0$, then g is a *non-empty* grid cell. If a non-empty grid cell g contains at least *minPts* points of P, then g is a *dense grid cell*. Otherwise, g is a *sparse grid cell*. Given a non-empty grid cell g, the neighbour grid cell set of g is defined as $\text{Nei}_g = \{g' \neq g \mid \exists p \in g, q \in g', s.t.\ \text{dist}(p, q) \leq \varepsilon\}$ (note here that p and q are not necessarily points in P). A grid cell, g', in Nei_g is called a neighbour grid cell of g. It can be verified that $|\text{Nei}_g| < (2\sqrt{d} + 3)^d \in O(1)$ [20, 22].

Our core point identification algorithm contains the following three steps:

Put Points into Grid Cells. Given a point p, the coordinates of the grid cell $g(p)$ which contains p can be calculated with the coordinates of p. Define $P_g = P \cap g$ as the set of all the points in P falling in g. By sorting all the points in P by the coordinates of the grid cells they belong to in lexicographical order, which can be achieved in $O(\log_s n)$ MPC rounds [24], all the points in P can be arranged in the machines with the following properties:

- Each machine stores points from consecutive (in the sorted order) non-empty grid cells. Those non-empty grid cells are called the *support grid cells* of the machine.
- The points in a same grid cell g are stored in either only one machine or a contiguous sequence of machines. Thus, the ID's of the machines that store P_g – the storage locations of the grid cell g – constitute an interval and can be represented by the left and right endpoints (i.e., the smallest and largest machine ID's) of this interval in $O(1)$ words.
- The number of the support grid cells of each machine is bounded by $O(s)$. And the total non-empty grid cell count is bounded by $O(n)$.

Calculate Non-empty Neighbour Grid Cell Storage Locations. In this step, we aim to, for each non-empty grid cell, compute the locations of all its *non-empty* neighbour grid cells. Since the size of Nei_g is bounded by $O((2\sqrt{d}+3)^d) \in O(1)$, it takes $O(1)$ space to store the location intervals of its all (possibly empty) neighbour grid cells.

At high level, our algorithm (Algorithm 4 in Appendix A) takes four *super rounds* to compute non-empty neighbour grid cell storage locations, as follows:

- Super Round 1: For each support grid cell g_j in a machine M_i, for each of its possible neighbours g', our algorithm builds a 4-tuple $(g_j.\text{coord}, M_i, g'.\text{coord}, g'.\text{loc})$, where $g.\text{coord}$ denotes the coordinates of grid cell g, M_i denotes the machine ID and $g'.\text{loc}$ term is a placeholder with value NULL at the current stage and will be filled with the storage location of g' if g' is non-empty. Next, these 4-tuples are sorted by the third term, $g'.\text{coord}$, across all machines. Therefore, at the end of this super round, each machine stores a subset of these 4-tuples in its memory.
- Super Round 2: Each machine M broadcasts the minimal and maximal $g'.\text{coord}$ value of the 4-tuples it stores, denoted by $[M.\min(g'.\text{coord}), M.\max(g'.\text{coord})]$. As a result, at the end of this super round, each machine, M_i, has the information of the $g'.\text{coord}$ value range of each other.
- Super Round 3: Each machine M_i sends a 2-tuple $(g_j.\text{coord}, g_j.\text{loc})$, for each of its support grid cells g_j, to those machines M having $g_j.\text{coord} \in [M.\min(g'.\text{coord}), M.\max(g'.\text{coord})]$. Specifically, the second term of the 2-tuple, $g_j.\text{loc}$, is the storage location of g_j, which was *computed* in the previous "put points into grid cells" phase.
- Super Round 4: After receiving all $g_j.\text{loc} \in [M.\min(g'.\text{coord}), M.\max(g'.\text{coord})]$ in Super Round 3, the previous placeholder, the forth term $g'.\text{loc}$, in each 4-tuple stored in machine M now can be filled properly in M's local memory. Each of the resultant 4-tuples $(g_j.\text{coord}, M_i, g'.\text{coord}, g'.\text{loc})$ are then sent back to machine M_i accordingly.

▶ **Lemma 21 ([*]).** *The above algorithm, i.e., Algorithm 4, takes $O(1)$ rounds. In each round, the amount of data sent and received by each machine is bounded by $O(s)$.*

Output the Point Labels. Algorithm 5 in Appendix A gives a detailed implementation. A running example of Algorithm 5 is shown in Figure 3. This algorithm labels each point in P as whether or not it is a core point. If a grid cell g is dense, i.e., $|g| \geq \mathit{minPts}$, all of the points in P_g are labelled as core points. For each point p in each sparse grid cell $g(p)$ (i.e., with $|g(p)| < \mathit{minPts}$) in each machine M, we can count the number of the points of P in the hyper ball centred at p with radius ε, denoted by $B(p, \varepsilon)$. Observe that a point q falls in $B(p, \varepsilon)$ only if either p and q are in the same grid cell or $g(q)$ is a neighbour grid cell of $g(p)$. Our algorithm, thus, sends p to all the storage locations of (i.e., the machines storing) the neighbour grid cells of $g(p)$. In the local memory of each of those machines, the distances between p and all those points in the neighbour grid cells can be computed. And then, the numbers of points falling in $B(p, \varepsilon)$ in those machines are sent back to machine M (where p is stored). Therefore, the points p in sparse cells can be labelled accordingly.

▶ **Lemma 22 ([*]).** *The above algorithm, i.e., Algorithm 5, performs two MPC rounds. In each round, the amount of data sent and received by each machine is bounded by $O(s)$.*

▶ **Lemma 23** (Core Point Identification **[*]**). *Consider a set P of n points; given a radius parameter $\varepsilon \geq 0$ and a core point threshold, $\mathit{minPts} \geq 1$, which is a constant integer, there exists an MPC algorithm with local memory $\Theta(s) \subseteq \Theta(n^\alpha)$ per machine, for arbitrary constant $\alpha \in (\frac{1}{2}, 1)$, which identifies all the core points in $O(\frac{1}{\alpha})$ rounds.*

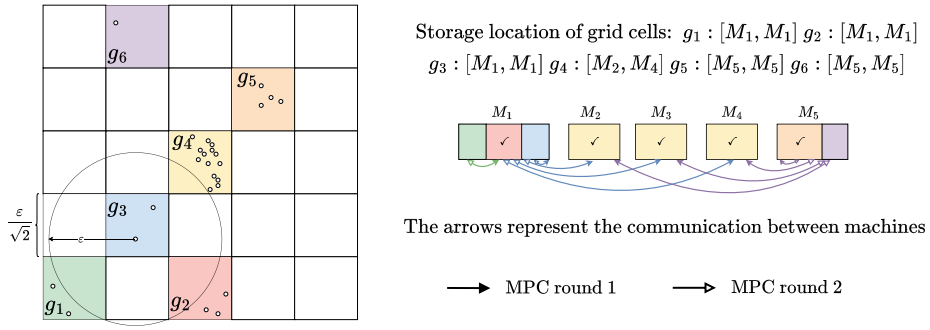

Figure 3 A running example of Algorithm 5 with $minPts = 3$. The input points are in six non-empty grid cells stored across five machines $M_1, \ldots, M_5$. Specifically, g_2, g_4, g_5 are dense grid cells and the points in them are core points, while g_1, g_3, g_6 are sparse grid cells. To label the points in these sparse grid cells, Algorithm 5 performs in two MPC rounds. Take g_3 stored in M_1 as an example; the neighbour grid cell set of g_3 is $\text{Nei}_{g_3} = \{g_1, g_2, g_4\}$, and the storage locations of them are: M_1 for both g_1 and g_2, and M_2, M_3, M_4 for g_4. The communications between M_1 and M_1 (i.e., local computation) and M_2, M_3, M_4 are incurred, which are shown by the blue arrows in the figure.

4.2.2 Compute Primitive Clusters

Our algorithm works as follows:

- construct an implicit (d, c)-grid graph $G' = (V', \mathcal{E}')$:
  - initialize $V' \leftarrow \varnothing$; impose a grid on P_{core} with side length of $\frac{1}{2\sqrt{d}}\rho\varepsilon$;
  - for each non-empty grid cell g, add the most *bottom-left* corner point t of g to V';
  - set $\mathcal{E}'$ to consider those edges (t_1, t_2) with $\text{dist}(t_1, t_2) \leq (1 + \frac{1}{2}\rho) \cdot \varepsilon$ only and returns $\text{dist}(t_1, t_2)$ as its weight; as a result, G' is c-penetration with $c = \frac{(1+\frac{1}{2}\rho)\cdot\varepsilon}{\frac{1}{2\sqrt{d}}\rho\varepsilon} \in O(1)$;
- invoke our MPC algorithm in Theorem 16 on the implicit (d, c)-grid graph $G' = (V', \mathcal{E}')$ to compute an MSF of G' and associate the connected component ID to each point $t \in V'$;
- for each point $t \in V'$, assign t's connected component ID to each core point in the grid cell which t represents;
- group all the core points in P_{core} by their connected component ID's;
- return each group as a primitive cluster; denote the set of these primitive clusters by $\mathcal{C}$;

▶ **Lemma 24 ([*]).** *The above primitive cluster computing algorithm returns legal ρ-approximate primitive clusters in $O(1)$ rounds with high probability.*

5 Conclusion

In this paper, we study the problems of grid graph connectivity, approximate EMST and approximate DBSCAN in the MPC model with strictly sub-linear local memory space per machine. We show Las Vegas algorithms (succeeding with high probability), which can be derandomized, for solving these problems in $O(1)$ rounds. Due to the importance of the problems studied in this paper, we believe that our $O(1)$-round MPC algorithms and then pseudo s-separator technique will be of independent interests for solving other related problems in the MPC model.

References

1 Pankaj K Agarwal, Herbert Edelsbrunner, Otfried Schwarzkopf, and Emo Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. In *Proceedings of the sixth annual symposium on Computational geometry*, pages 203–210, 1990. doi:10.1145/98524.98567.

2 Pankaj K. Agarwal, Kyle Fox, Kamesh Munagala, and Abhinandan Nath. Parallel algorithms for constructing range and nearest-neighbor searching data structures. In *Proceedings of the 35th ACM PODS 2016*, pages 429–440. ACM, 2016. doi:10.1145/2902251.2902303.

3 Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, 1988. doi:10.1145/48529.48535.

4 Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, 1998.

5 Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 574–583. ACM, 2014. doi:10.1145/2591796.2591805.

6 Alexandr Andoni, Zhao Song, Clifford Stein, Zhengyu Wang, and Peilin Zhong. Parallel graph connectivity in log diameter rounds. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*, 2018. doi:10.1109/FOCS.2018.00070.

7 Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998. doi:10.1145/293347.293348.

8 Sepehr Assadi, Xiaorui Sun, and Omri Weinstein. Massively parallel algorithms for finding well-connected components in sparse graphs. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019*, 2019.

9 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. *Journal of the ACM (JACM)*, 64(6):1–58, 2017. doi:10.1145/3125644.

10 Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, and Vahab Mirrokni. Near-optimal massively parallel graph connectivity. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1615–1636. IEEE, 2019. doi:10.1109/FOCS.2019.00095.

11 Patrick Burger and Hans-Joachim Wuensche. Fast multi-pass 3d point segmentation based on a structured mesh graph for ground vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 2150–2156. IEEE, 2018. doi:10.1109/IVS.2018.8500552.

12 Brent N Clark, Charles J Colbourn, and David S Johnson. Unit disk graphs. *Discrete mathematics*, 86(1-3):165–177, 1990. doi:10.1016/0012-365X(90)90358-O.

13 Sam Coy and Artur Czumaj. Deterministic massively parallel connectivity. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 162–175. ACM, 2022. doi:10.1145/3519935.3520055.

14 Sam Coy, Artur Czumaj, and Gopinath Mishra. On parallel k-center clustering. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2023, Orlando, FL, USA, June 17-19, 2023*, pages 65–75. ACM, 2023. doi:10.1145/3558481.3591075.

15 Artur Czumaj, Guichen Gao, Shaofeng H.-C. Jiang, Robert Krauthgamer, and Pavel Veselý. Fully scalable MPC algorithms for clustering in high dimension. *CoRR*, abs/2307.07848, 2023. doi:10.48550/arXiv.2307.07848.

16 Artur Czumaj, Christiane Lammersen, Morteza Monemizadeh, and Christian Sohler. $(1+\varepsilon)$-approximation for facility location in data streams. In *Proceedings of SODA 2013*, 2013.

17 Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008. doi:10.1145/1327452.1327492.

18 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. arXiv:1810.04805.

19 Hu Ding and Jinhui Xu. Solving the chromatic cone clustering problem via minimum spanning sphere. In *International Colloquium on Automata, Languages, and Programming*, pages 773–784. Springer, 2011. doi:10.1007/978-3-642-22006-7_65.

20 Junhao Gan and Yufei Tao. Dbscan revisited: Mis-claim, un-fixability, and approximation. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 519–530, 2015. doi:10.1145/2723372.2737792.

21 Junhao Gan and Yufei Tao. On the hardness and approximation of euclidean dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):1–45, 2017. doi:10.1145/3083897.

22 Junhao Gan and Yufei Tao. An i/o-efficient algorithm for computing vertex separators on multi-dimensional grid graphs and its applications. *J. Graph Algorithms Appl. (JGAA)*, 22(2):297–327, 2018. doi:10.7155/JGAA.00471.

23 Junhao Gan, Anthony Wirth, and Zhuo Zhang. $o(1)$-round mpc algorithms for multi-dimensional grid graph connectivity, emst and dbscan, 2025. arXiv:2501.12044.

24 Michael T Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In *International Symposium on Algorithms and Computation*, pages 374–383. Springer, 2011. doi:10.1007/978-3-642-25591-5_39.

25 Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016. doi:10.1109/CVPR.2016.90.

26 Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the thirty-sixth annual ACM Symposium on Theory of Computing*, pages 373–380, 2004. doi:10.1145/1007352.1007413.

27 Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2007 EuroSys Conference, Lisbon, Portugal, March 21-23, 2007*, pages 59–72. ACM, 2007. doi:10.1145/1272996.1273005.

28 Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, 2010. doi:10.1137/1.9781611973075.76.

29 Yi Li, Philip M. Long, and Aravind Srinivasan. Improved bounds on the sample complexity of learning. *J. Comput. Syst. Sci.*, 62(3):516–527, 2001. doi:10.1006/JCSS.2000.1741.

30 Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013. URL: http://arxiv.org/abs/1301.3781.

31 Morteza Monemizadeh. Facility location in the sublinear geometric model. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2023)*, 2023.

32 Danupon Nanongkai and Michele Scquizzato. Equivalence classes and conditional hardness in massively parallel computations. *Distributed computing*, 35(2):165–183, 2022. doi:10.1007/S00446-021-00418-2.

33 Abhinandan Nath, Kyle Fox, Pankaj K Agarwal, and Kamesh Munagala. Massively parallel algorithms for computing tin dems and contour trees for large terrains. In *Proceedings of the 24th ACM SIGSPATIAL*, 2016.

34 Tom White. *Hadoop - The Definitive Guide: Storage and Analysis at Internet Scale (3. ed., revised and updated)*. O'Reilly, 2012. URL: http://www.oreilly.de/catalog/9781449311520/index.html.

35 Jie Xue. Colored range closest-pair problem under general distance functions. In *Proceedings of ACM-SIAM SODA*, pages 373–390. SIAM, 2019. doi:10.1137/1.9781611975482.24.

36 Grigory Yaroslavtsev and Adithya Vadapalli. Massively parallel algorithms and hardness for single-linkage clustering under lp-distances. In *35th International Conference on Machine Learning (ICML'18)*, 2018.

37 Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, Boston, MA, USA, June 22, 2010*, 2010. URL: https://www.usenix.org/conference/hotcloud-10/spark-cluster-computing-working-sets.

38 Yan Zhou, Oleksandr Grygorash, and Thomas F Hain. Clustering with minimum spanning trees. *International Journal on Artificial Intelligence Tools*, 20(01):139–177, 2011. doi:10.1142/S0218213011000061.

A Appendix

Here, we include pseudocode for the main algorithms in our paper.

Algorithm 3 MSF Algorithm.

Input: implicit (d, c)-grid graph $G = (V, \mathcal{E})$ with $c \in [1, s^{\frac{1}{d^3}}]$
Output: an MSF of G

compute the pseudo s-separator with the algorithm in Section 2.2;
process each extended sub graph G_i^+ in their machine:
  compute the MSF of G_i^+, denote the edge set as T_i;
  compress T_i into an edge set E_i' as described in Algorithm 2;
  //each edge e in E_i' corresponds to a path in T_i and the heaviest edge weight in that path is called the *bottleneck edge of* e ;
  set the weight of edges in E_i' the same as its bottleneck edge weight;
  send E_i' to machine M_0;
M_0 run:
  build a graph based on all of the received edges $E' = \cup E_i'$;
  solve the MSF problem on E', denoted as T';
  assign to each edge in E' a label to indicate whether it is in T';
  send E_i' back to the corresponding machine along with the label;
process each extended sub graph G_i^+ in their machine:
  for each edge in E_i' labeled as negative, i.e., not included in T', (conceptually) remove its bottleneck edge from G_i^+ ;
  //let $G_i^{+'}$ denote the extended sub graph after conceptually removing those edges;
  calculate the MSF of $G_i^{+'}$, denote the result as $\mathcal{T}_i$;
//the union of all $\mathcal{T}_i$ is the MSF of G;

Algorithm 4 Location of Non-empty Neighbor Grid Cell.

Input: A set of m machines $(M_1, \ldots, M_m)$, each machine M_i contains points from $k^{(i)}$ grid cells $(g_1^{(i)}, \cdots, g_{k^{(i)}}^{(i)})$, the storage location of the grid cell g.loc

Output: Given a non-empty grid cell g, calculate the storage location of every $g' \in \text{Nei}_g$

for *machine* M_i, $i \in [1, m]$ *simultaneously* **do**
  for *each support grid cell* g_j, $j \in [1, k^{(i)}]$ **do**
    build 4-tuples $(g_j.\text{coord}, M_i, g'.\text{coord}, g'.\text{loc})$ for each $g' \in \text{Nei}_{gj}$;
    //the fourth term is a placeholder with value NULL now
invoke a sorting algorithm for all of the 4-tuples based on the g'.coord;
for *machine* M_i, $i \in [1, m]$ *simultaneously* **do**
  broadcast the minimal and maximal g'.coord ;
  //each machine also receives that information during the communication phase;
  for *each grid cell* g_j, $j \in [1, k^{(i)}]$ **do**
    send the 2-tuple $(g_j.\text{coord}, g_j.\text{loc})$ to the machines M s.t. $g_j.\text{coord} \in [M.\min(g'.\text{coord}), M.\max(g'.\text{coord})]$;
//each machine also receives the 2-tuple during the communication phase;
merge the 2-tuple with 4-tuple to fill in the placeholder;
for *each 4-tuple with* $g'.loc! = NULL$ **do**
  send the 4-tuple back to M_i;
//each machine receive the backward 4-tuple during the communication phase;

Algorithm 5 Output the Point Label.

Input: A set of m machines $(M_1, \cdots, M_m)$, machine M_i contains points from $k^{(i)}$ grid cells $(g_1^{(i)}, \cdots, g_{k^{(i)}}^{(i)})$, the storage location of the grid cell g.loc

Output: For each point p, assign to it a label as core or non-core

for *machine* M_i, $i \in [1, m]$ *simultaneously* **do**
  for *each grid cell* g_j, $j \in [1, k^{(i)}]$ **do**
    if g_j *is a dense grid cell* **then**
      p.type $\leftarrow$ core;
    else
      for *each point* p *in* g_j **do**
        for *each non-empty neighbor* g' *of* g_j **do**
          send p.coord to all machines in g'.loc;
//each machine receives a set of points P_{in};
for *machine* M_i, $i \in [1, m]$ *simultaneously* **do**
  for *each point* p *in* P_{in} **do**
    count the point number that falls in $B(p, \varepsilon)$;
    send that number back to where point p is from;
  //each machine receives the number;
  for *each point* p *in sparse grid cell* **do**
    count the total point number that falls in $B(p, \varepsilon)$;
    identify the type of p