Massively Parallel Algorithms for the Stochastic Block Model

Zelin Li
School of Computer Science and Technology, University of Science and Technology of China, Hefei, China

Pan Peng
School of Computer Science and Technology, University of Science and Technology of China, Hefei, China

Xianbin Zhu
Department of Computer Science, City University of Hong Kong, Hong Kong, China

Abstract

Learning the community structure of a large-scale graph is a fundamental problem in machine learning, computer science and statistics. Among others, the Stochastic Block Model (SBM) serves a canonical model for community detection and clustering, and the Massively Parallel Computation (MPC) model is a mathematical abstraction of real-world parallel computing systems, which provides a powerful computational framework for handling large-scale datasets. We study the problem of exactly recovering the communities in a graph generated from the SBM in the MPC model. Specifically, given kn vertices that are partitioned into k equal-sized clusters (i.e., each has size n), a graph on these kn vertices is randomly generated such that each pair of vertices is connected with probability p if they are in the same cluster and with probability q if not, where $p > q > 0$.

We give MPC algorithms for the SBM in the (very general) *s-space MPC model*, where each machine is guaranteed to have memory $s = \Omega(\log n)$. Under the condition that[1] $\frac{p-q}{\sqrt{p}} \geq \tilde{\Omega}(k^{\frac{1}{2}} n^{-\frac{1}{2}+\frac{1}{2(r-1)}})$ for any integer $r \in [3, O(\log n)]$, our first algorithm exactly recovers all the k clusters in $O(kr \log_s n)$ rounds using $\tilde{O}(m)$ total space, or in $O(r \log_s n)$ rounds using $\tilde{O}(km)$ total space. If $\frac{p-q}{\sqrt{p}} \geq \tilde{\Omega}(k^{\frac{3}{4}} n^{-\frac{1}{4}})$, our second algorithm achieves $O(\log_s n)$ rounds and $\tilde{O}(m)$ total space complexity. Both algorithms significantly improve upon a recent result of Cohen-Addad et al. [PODC'22], who gave algorithms that only work in the *sublinear space MPC model*, where each machine has local memory $s = O(n^\delta)$ for some constant $\delta > 0$, with a much stronger condition on p, q, k. Our algorithms are based on collecting the r-step neighborhood of each vertex and comparing the difference of some statistical information generated from the local neighborhoods for each pair of vertices. To implement the clustering algorithms in parallel, we present efficient approaches for implementing some basic graph operations in the s-space MPC model.

2012 ACM Subject Classification Theory of computation → Massively parallel algorithms

Keywords and phrases Massively Parallel Computation, Stochastic Block Model, Graph Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2023.78

Related Version *Full Version*: https://arxiv.org/abs/2307.00530

Funding *Zelin Li and Pan Peng*: Supported in part by NSFC grant 62272431 and "the Fundamental Research Funds for the Central Universities".
Xianbin Zhu: Supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 11213620].

Acknowledgements We would like to thank the anonymous reviewers for their detailed comments.

[1] $\tilde{\Omega}(\cdot)$ hides $\mathrm{poly}(\log kn)$ factors.

1 Introduction

Graph clustering is a fundamental task in machine learning, computer science and statistics. In this task, given a graph that may represent a social/information/biological network, the goal is to partition its vertex set into a few maximal subsets (called *clusters* or *communities*) of similar vertices. Depending on the context, a cluster may correspond to a social group of people with the same hobbies, a group of web-pages with similar contents or a set of proteins that interact very frequently. Intuitively, in a good clustering of a graph, there are few edges between different clusters while there are relatively many edges inside each cluster. There is no unified formalization on the notions of graph clustering and clusters. Here we focus on a natural and widely-used model for graph clustering, the *stochastic block model (SBM)*. In the SBM, we are given a set V of $N = kn$ vertices such that there is a hidden partition of V with $V = \bigcup_{i=1}^{k} V_i$, $V_i \cap V_j = \emptyset$ for any $1 \leq i < j \leq k$, where each set V_i is called a *cluster* (or *community*). For simplicity, we assume that each cluster has an equal size, i.e., $|V_i| = n$. We say a graph $G = (V, E)$ is generated from the SBM with parameters n, p, q, k, abbreviated as $\mathrm{SBM}(n, p, q, k)$, if for any two vertices u, v that belong to the same cluster, the edge (u, v) appears in G with probability p; for any two vertices u, v that belong to two different clusters, the edge (u, v) appears with probability q, where $0 < q < p < 1$.

Thanks to its simplicity and its ability in explaining the community structures in real world data, the SBM has been extensively studied in the computer science literature. Most previous work has been focusing on algorithms that work on a single machine, with the goal of extracting the communities with the optimal (computational and/or statistical) trade-offs between parameters n, p, q, k, for different types of recoveries (i.e., exact, weak, and partial recovery). Significant progress has been made on such algorithms (and their limitations) in the past decades (see the survey [1]). However, most of these algorithms are essentially sequential and cannot be adapted to the parallel or distributed environment, which is unsatisfactory as modern graphs are becoming massive and most of them cannot be fitted into the main memory of a single machine.

We study the problem of exactly recovering communities of a graph from the SBM in the massively parallel computation (MPC) model [24, 22, 5], which is a mathematical abstraction of modern frameworks of real-world parallel computing systems like MapReduce [19], Hadoop [28], Spark [29] and Dryad [23]. In this model, there are M machines that communicate in synchronous rounds, where the local memory of each machine is limited to s words, each of $O(\log n)$ bits. A word is enough to store a node or a machine identifier from a polynomial (in n) domain. Communication is the largest bottleneck in the MPC model. Take the graph problem as an example. The edges of the input graph are arbitrarily distributed across the M machines initially. Ideally, we would like to use minimal number of rounds of computation while using small (say sublinear) space per machine and small total space (i.e., the sum of space used by all machines).

Recently, Cohen-Addad et al. [16] gave two algorithms Majority and Louvain that recover the communities in a graph generated from $\mathrm{SBM}(n, p, q, k)$ when $\frac{p-q}{\sqrt{p}} \geq \Omega(n^{-\frac{1}{4}+\varepsilon})$ and k is constant. They work in $O(\frac{1}{\varepsilon \cdot \delta})$ rounds in the *sublinear space* MPC model, i.e., each machine has local memory $s = O(n^{\delta})$, for any constant $\delta > 0$. Their algorithms and analysis improve upon previous sequential versions of Majority and Louvain given by Cohen-Addad et al [12]. Note that for any sequential algorithm, it is known that $\frac{p-q}{\sqrt{p}} = \Omega(\sqrt{\frac{\log n}{n}})$ is necessary for exact recovery even for $k = 2$ [2]; there exist spectral algorithms and SDP-based algorithms that find all clusters and achieve this parameter threshold [1]. Therefore, it is natural to ask *if one can obtain a round-efficient MPC algorithm in the sublinear space model with roughly the same parameter threshold.*

In this paper, we consider a more general setting that we call *the s-space MPC model* in which the local memory s is only guaranteed to satisfy that $s = \Omega(\log n)$. Nowadays, the growth rate of data volume far exceeds the growth rate of machine hardware storage and it is likely that we need much more machines to analyze large-scale data. Furthermore, the problem of clustering of data points from some metric space on such a model has recently received increasing interest [8, 20, 4, 14, 17, 18] (see also Section 1.3), partly due to the fact that in some scenarios, the number of clusters k is too large such that even just storing k representatives of all the clusters is not possible in a single machine. Note that this model is more difficult to handle than the sublinear space model, and we need to carefully partition the data across machines so that different machines work in different "regions of space" to get a good tradeoff between communication and the used space. Here, we are interested in the question *whether we can obtain an SBM clustering algorithm in the s-space MPC model with good tradeoffs between communication, space and SBM parameters.*

1.1 Our Results

We give clustering algorithms for the SBM that work in the s-space MPC model where the local memory s of each machine is only guaranteed to satisfy that $s = \Omega(\log n)$. Let $m = \Theta(kn^2p + k^2n^2q)$ denote the total number of edges of the graph (our conditions always imply that $p \geq \Omega(\frac{\log n}{n})$ and $k \leq n$). We use "with high probability" to denote "with probability at least $1 - O(n^{-1})$".

Our first algorithm has the following performance guarantee.

▶ **Theorem 1.** *Let r be any integer such that $3 \leq r \leq O(\log n)$. Let $p, q \leq 0.75$ be parameters such that $\max\{p(1-p), q(1-q)\} \geq C_0 \log n/n$ where $C_0 > 0$ is some constant. Suppose that $\frac{p-q}{\sqrt{p}} \geq \Omega\left(k^{\frac{1}{2}} n^{-\frac{1}{2}+\frac{1}{2(r-1)}} \log^7(kn)\right)$. Let G be a random graph generated from SBM(n, p, q, k). Then there exists an algorithm in the s-space MPC model that outputs k clusters in $O(kr \log_s n)$ rounds with high probability where each machine has $s = \Omega(\log n)$ memory. The total space used by the algorithm is $\tilde{O}(m)$.*

We note that the round complexity can be improved to be $O(r \log_s n)$ at the cost of increasing the total space by a k factor, which is formalized in the following theorem.

▶ **Theorem 2.** *Under the same condition in Theorem 1, there exists an algorithm that outputs k clusters in $O(r \log_s n)$ rounds where each machine has $s = \Omega(\log n)$ memory with high probability and uses $\tilde{O}(km)$ total space.*

Note that for any integer constant $3 \leq r \leq o(\log n)$ and any $k \leq \text{poly}(\log n)$, the round complexity of the above algorithm is $O(\log_s n)$ while the total space is $\tilde{O}(m)$. When $r = \Theta(\log n)$, then the recovery condition becomes $\frac{p-q}{\sqrt{p}} \geq \tilde{\Omega}(\sqrt{\frac{k}{n}})$, which almost matches the statistical limit in the sequential setting up to logarithmic terms [2]. In this case, our algorithm has round complexity $O(\log n \log_s n)$ for any $k \leq \text{poly}(\log n)$ in the s-space MPC model.

When the gap between p, q is sufficiently large, we can achieve $O(\log_s n)$ rounds using $\tilde{O}(m)$ total space, i.e., both the round complexity and the total space complexity are independent of the number k of clusters. Formally, we have the following theorem.

▶ **Theorem 3.** *Given a random graph G from SBM(n, p, q, k) with $\frac{p-q}{\sqrt{p}} \geq \Omega\left(k^{\frac{3}{4}} n^{-\frac{1}{4}} (\log n)^{\frac{1}{4}}\right)$, there exists an algorithm in the s-space MPC model that can output k hidden clusters within $O(\log_s n)$ rounds with high probability, where $s = \Omega(\log n)$, and uses $\tilde{O}(m)$ total space.*

We note that all the algorithms in Theorem 1, 2 and 3 significantly improve the results of [16], of which the algorithms only work in the sublinear space MPC model, i.e., $s = O(n^{\delta})$ for some constant $\delta > 0$, and finish in $O(\frac{1}{\delta\varepsilon})$ rounds, assuming that $\frac{p-q}{\sqrt{p}} \geq n^{-1/4+\varepsilon}$ and k is a constant. In both sublinear space and s-space models, our algorithms work for a much wider class of SBM graphs (i.e., the requirement on the conditions of p, q, k are much weaker) than those in [16]. Furthermore, even for the same regime of parameters, our algorithms have better round complexity. For example, in the sublinear space MPC model, our round complexity (from Theorem 3) is $O(1/\delta)$ under the condition that $\frac{p-q}{\sqrt{p}} \geq \Omega(n^{-\frac{1}{4}}(\log n)^{\frac{1}{4}})$ and k is constant, while the algorithms in [16] have round complexity $O(\frac{\log n}{\delta \log\log n})$ under the same condition[2].

Our algorithms are quite different from those in [16], in which the algorithms are based on the local-search methods and proceed in rounds by updating the so-called *swap values* for each node to decide where to move the node. Our algorithms are based on collecting the r-step neighborhood of each vertex and comparing the difference of some statistical information generated from the local neighborhoods for each pair of vertices.

To implement the above MPC algorithms, we give new algorithms of some basic graphs operations in the s-space MPC model in Section 3, including **RandomSet** (for randomly sampling a set), **ReorganizeNBR** that is for organizing the neighborhood of any two nodes u, v in a set so that they are "aligned", i.e., the i-th byte of u (or v) indicates whether the i-th node is the neighbor of u (or v). We believe these results will be useful as basic tools in designing algorithms for other problems in the s-space MPC model.

1.2 Our Techniques

Our MPC algorithms are based on two simple sequential algorithms. We first describe our first algorithm given in Theorem 3. It is based on the observation that if $\frac{p-q}{\sqrt{p}} \geq \tilde{\Omega}(k^{\frac{3}{4}} n^{-\frac{1}{4}})$, then the number of common neighbors of any two vertices can be used to distinguish if they belong to the same cluster or not. That is, if u, v belong to the same cluster, then the number of their common neighbors is above some threshold Δ; otherwise, the number of common neighbors is smaller than Δ. Let $N(v)$ denote the set of all the neighbors of v. We further note that to get k clusters of V, it is not necessary to compute $|N(u) \cap N(v)|$ for *all pairs* of u, v in V, which may cause too much communication for MPC implementation. Instead, we first randomly sample a small set S' with $|S'| = \Theta(k \log n)$. Then we find k representatives of the hidden clusters from S' by computing $|N(u) \cap N(v)|$ for all pairs of u, v in S' and update S' to be the set of k representatives. Then we sample independently another small set S of vertices, and find k sub-clusters from S by computing $|N(u) \cap N(v)|$ for $u \in S$ and $v \in S'$. (A set $T \subseteq S$ is called a *sub-cluster* of some cluster V_i if $T \subseteq V_i$.) Based on the k sub-clusters obtained from S, we can find all the hidden clusters $V_1, \ldots, V_k$ putting any vertex $v \in V \setminus S$ to the sub-cluster that contains the most number of neighbors of v.

There are several challenges to implementing the above algorithm in the s-space MPC model in which the local memory only satisfies that $s = \Omega(\log n)$. Note that in this model, even just to compute the number of common neighbors $|N(u) \cap N(v)|$ for any *fixed pair* u, v in a few parallel rounds (say $O(\log_s n)$ rounds) is non-trivial. The reason is that the neighborhoods $N(u), N(v)$ can be much larger than s and some neighborhoods will be used too many times which leads to large round complexity. To efficiently compute $|N(u) \cap N(v)|$ for $u \in S$ and $v \in S'$, we first show how to reorganize $N(u)$ and $N(v)$ for $u \in S$ and $v \in S'$

[2] This can be seen by setting $\frac{1}{\varepsilon} = \Theta(\frac{\log n}{\log\log n})$ in [16].

so that each byte of $N(u)$ and $N(v)$ for any two nodes aligned; then we can show how to compute $|N(u) \cap N(v)|$ in parallel efficiently by appropriately making some copies of $N(u)$ and $N(v)$. For these tasks, we give detailed MPC implementations of some basic operations, e.g., a procedure for copying neighbors of some carefully chosen nodes and aligning their neighbors while using no more than $\tilde{O}(m)$ total space.

Our MPC algorithms from Theorem 1 and 2 are based on a recent sequential algorithm given in [25]. Roughly speaking, one can use the power iterations of some matrix $B = A - q \cdot J$ to find the corresponding clusters, where A is the adjacency matrix of the graph and J is the all-1 matrix. It is shown that with high probability, the ℓ_2-norm of $B_u^r - B_v^r$ is relatively small, if u, v belong to the same cluster; and is large, otherwise. Here B_u^r is the row corresponding to vertex u in the matrix B^r. We show that in order to compute $\|B_u^r - B_v^r\|_2$, it suffices to compute the expressions $\mathbf{1}_x^T (A - qJ)^{2r} \mathbf{1}_y$ for all $x, y \in \{u, v\}$. To do so, we expand the above expression so that we get a sum of terms, each being a vector-matrix-vector multiplication. Then we give a combinatorial explanation of each term, and then calculate it in parallel efficiently based on some basic graph operations in the s-space model.

1.3 Related Work

There is a line of research on metric clustering in the MPC model. In this setting, the input is a set of data points from some metric space (e.g., Euclidean space), and the goal is to find k representative centers, such that some objective function (e.g. the cost functions of k-means, k-median and k-center) is minimized (e.g., [8]). Bhaskara and Wijewardena [8] developed an algorithm that outputs $O(k \log k \log n)$ centers whose cost is within a factor of $O((\log n \log\log n)^2)$ of the optimal k-means (or k-median) clustering, using a memory of $s \in \Omega(d \log n)$ per machine and $O(\log_s n)$ parallel rounds. Note that this does not require $\Omega(k)$ memory per machine. Coy et al. [18] recently improved the approximation ratio of the algorithm for k-center in [8] to $O(\log^* n)$. Cohen-Addad et al. [17] gave a fully scalable $(1+\varepsilon)$-approximate k-means clustering algorithm when the instance exhibits a "ground-truth" clustering structure, captured by a notion of "$O(\alpha)$-perturbation resilient", and it uses $O(1)$ rounds and $O_{\varepsilon,d}(n^{1+1/\alpha^2+o(1)})$ total space with arbitrary memory per machine, where each data point is from $\mathbb{R}^d$.

Regarding the power method for SBM, Wang et al. [27] proposed an iterative algorithm that first employs the power method with a random starting point and then turns to a generalized power method that can find the communities in a finite number of iterations. Their algorithm runs in nearly linear time and can exactly recover the underlying communities at the information-theoretic limit. Cohen-Addad et al. [15] further gave a linear-time algorithm that recovers exactly the communities at the asymptotic information-theoretic threshold. Their algorithm is based on similar ideas as in [16], that is, given a partition, moving a vertex from one part to the part where it has most neighbors should somewhat improve the quality of the partition.

Correlation clustering has been studied under the MPC model. In this problem, a signed graph $G = (V, E, \sigma)$ is given as input, and the goal is to partition the vertex set into arbitrarily many clusters so that the disagreement of the corresponding clustering is minimized, where the disagreement is the number of edges that cross different clusters plus the number of non-adjacent pairs inside the clusters [9, 11, 26, 21, 10, 13, 3]. The state-of-the-art is a $(3+\varepsilon)$-approximation algorithm in $O(1/\varepsilon)$ rounds in the massively parallel computation (MPC) with sublinear space [7].

2 Preliminaries

Consider an undirected graph $G = (V, E)$ where V is the set of vertices, and E is the set of edges. We use n to denote the size of $|V|$ and m to denote the size of $|E|$. Each node in G has a unique ID from 1 to n. We use $\mathsf{ID}(u)$ to denote the ID for a node $u \in V$. We use $d(u)$ to denote the degree of $u \in V$. Let $N(u)$ denote the set of neighbors of a node $u \in V$. Given a vertex set $S \subset V$, we use $G[S]$ to denote the subgraph induced by vertices in S. In this paper, we abuse the use of node(s) and vertex(vertices). We use $[i]$ to denote $\{1, 2, \cdots i\}$. When nodes are active (inactive), they execute (do not execute) algorithms.

Chernoff Bound

Let $X_1, ..., X_n$ be independent binary random variables, and $X = \Sigma_{i=1}^{n} X_i$, and $\mu = \mathbb{E}[X]$. Then it holds that for all $\delta > 0$ that $\mathbb{P}[X \geq (1+\delta)\mu] \leq (\frac{e^{-\delta}}{(1+\delta)^{1+\delta}})^{\mu} \leq e^{-\min[\delta^2, \delta]\mu/3}$; For all $\delta \in (0, 1)$, $\mathbb{P}[X \leq (1-\delta)\mu] \leq (\frac{e^{\delta}}{(1-\delta)^{1-\delta}})^{\mu} \leq e^{-\delta^2\mu/2}$.

The MPC model

In this model, we assume that all data is arbitrarily distributed among some machines. Let N denote the total amount of data. Each machine has local memory s. In our settings, $s = \Omega(\log n)$. The sum of all local memory is $N = O((m+n)\text{poly}(\log n))$. The communication between any pair of two machines is synchronous, and the bandwidth is s words.

We ignore the cost of local communication and computation happening in each machine. As the description of the MPC model in the literature, we ignore some communication details among different machines and suppose that all machines are known to each other which means that any machine can send messages to another machine directly (even when the local memory is very small). For the problems in the MPC model, we aim to make the total number of communication rounds among machines as small as possible.

▶ **Definition 4** (separable function). *Let $f : 2^{\mathbb{R}} \to \mathbb{R}$ denote a set function. We say that f is separable if and only if for any set of reals A and for any $B \subseteq A$, we have $f(A) = f(f(B), f(A \setminus B))$* [3].

▶ **Lemma 5** ([6]). *Given an n-vertex graph, we have x_u for each node $u \in V$. If the function f is a separable function, then there exists an algorithm that computes $f(\{x_i \in N(u)\})$ for each $u \in V$ with high probability in the sublinear space MPC model in $O(1/\delta)$ rounds using $\tilde{O}(m)$ space where each machine has space $O(n^{\delta})$.*

In the s-space MPC model, we restate the following folklore lemma.

▶ **Lemma 6.** *Given an n-vertex graph, there exists an algorithm that makes each node $u \in V$ visit $N(u)$ in $O(\log_s n)$ rounds with high probability.*

The sorting algorithm is a very important black-box tool in the MPC model, which is stated as follows.

▶ **Theorem 7** ([22]). *Sorting can be solved in $O(\log_s n)$ rounds in the s-space MPC model.*

[3] For example, f can be a sum function.

Furthermore, it has been shown that indexing and prefix-sum operation can be performed in $O(\log_s n)$ rounds [22]. We refer to the INDEX Algorithm for solving indexing problems in the s-space MPC model and the SORTING Algorithm for solving sorting problems in the same model. Throughout the context, we will rely on the fundamental properties associated with the aforementioned operations, as well as Lemma 5 and Lemma 6 by default.

3 Implementing Basic Graph Operations in the s-space MPC Model

In this section, we present algorithms for several fundamental graph operations in the s-space model, which will be utilized in our MPC algorithms. To the best of our knowledge, most of these operations have not been previously implemented in the s-space MPC model. We denote the machines holding node x as M_x. (It is important to note that the MPC model follows an edge-partition model, which means that multiple machines may hold the same vertices). It is worth mentioning that in order to implement some of our proposed algorithms, we utilize previous algorithms for basic MPC operations, as demonstrated in the full version.

RandomSet

In the RandomSet problem, given an input value $X = \Omega(\log n)$, our goal is to output a random set $S = \{x_1, x_2, \ldots, x_{|S|}\}$ where each element x_i ($i \in [|S|]$) is selected uniformly and independently at random and $S' = \{(x, y)|x \in S \cap M_x, y = \mathrm{Ind}_S(x)\}$ where $|S| = \Theta(X)$, and $\mathrm{Ind}_S(x)$ is the index of of $x \in S$ in S. We use the algorithm RANDOMSET to solve the RandomSet problem.

▶ **Lemma 8.** *The RandomSet problem can be solved in the s-space MPC model in $O(\log_s n)$ rounds where $s = \Omega(\log n)$.*

ReorganizeNBR

The ReorganizeNBR problem involves an input set S, where the objective is to reorganize $N(u)$ for all $u \in S$ in a manner that *aligns* the bytes of $N(u)$ and $N(v)$ for any two nodes $u, v \in S$. Specifically, the i-th byte of $N(u)$ indicates whether the i-th node is a neighbor of u. The motivation behind the ReorganizeNBR problem is to efficiently compute $|N(u) \cap N(v)|$ for any pair of nodes u and v in S (refer to Figure 1 for illustration). We utilize the REORGANIZENBR algorithm to address this problem.

Now, we show how to reorganize a graph in MPC model where each machine has memory $s = \Omega(\log n)$. We say a subgraph H is a randomly sampled subgraph if nodes in V_H are randomly sampled, and H is the set of edges and vertices constructed from picking V_H with their incident edges. We use $H = (V_H, E_H)$ to denote such a random sampled graph.

▶ **Lemma 9.** *Given a graph $G = (V, E)$ with $m = \Theta(n^{c+1})$, where m is the number of edges and n is the number of vertices, and $c \in (0, 1]$ is some positive constant, for a given randomly sampled subgraph $H = (V_H, E_H)$ satisfying $|V_H| \leq \tilde{O}(m/n)$ where V_H is constructed by RANDOMSET, there exists an algorithm that can reorganize $N(u)$ for each $u \in V_H$ in constant rounds, with total space complexity $\tilde{O}(m)$ in the MPC model where each machine has a memory of $s = \Omega(\log n)$.*

CopyNBR($\mathcal{S}, t$)

Suppose we have $\mathcal{S} = \{N(v_1), \ldots, N(v_x)\}$ stored in machines where $v_i \in S$, $i \in [x]$ and S is a random set created by RANDOMSET. We will create $N(v_i)_1, \ldots, N(v_i)_t$ for each $N(v_i)$ where $i \in [x]$. The goal is to make t copies of $N(u)$ for each $u \in S$ such that we can execute

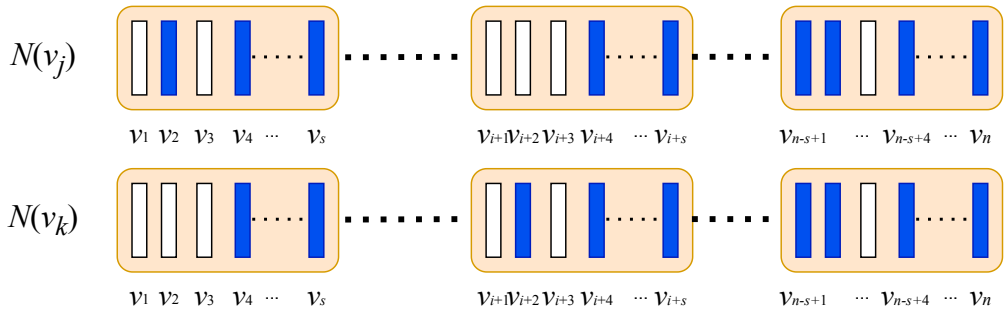

Figure 1 Align Operation. The first row represents $N(v_j)$, the blue rectangle (treated as "1") indicates that the corresponding vertex $v_i \in N(v_j)$, and the white rectangle (treated as "0") indicates that the vertex $v_i \notin N(v_j)$. Once $N(v_j)$ and $N(v_k)$ are encoded as such bit strings, we can compare the strings simultaneously to compute their common neighbors.

other algorithms in parallel. In our setup, each $N(u)$ where $u \in S$ is organized in a collection of consecutive machines. To solve this problem, we employ the CopyNBR$(\mathcal{S}, t)$ algorithm. Upon executing CopyNBR$(\mathcal{S}, t)$, all t copies of $\mathcal{S}$ are stored in consecutive machines.

▶ **Lemma 10.** *The CopyNBR$(\mathcal{S}, t)$ problem can be solved in $O(\log_s n)$ rounds in the s-space MPC model where $s = \Omega(\log n)$ and t is a parameter satisfying $n|S|t \leq \tilde{O}(m)$.*

EvenCluster

Consider a set S comprising nodes labeled from 1 to k. The objective is to ensure that the number of nodes with labels in S is even. To achieve this, we employ the EvenCluster algorithm, designed specifically to solve this problem.

▶ **Lemma 11.** *In the MPC model with each machine's memory $s = \Omega(\log n)$, there exists an algorithm that can output $S' \subseteq S$ within $O(\log_s n)$ rounds, such that each label in S' is associated with the same number of nodes with that label.*

RepresentativeK(S)

In the RepresentativeK(S) problem, the input is a set $S = S_1 \cup \cdots \cup S_k$ of nodes with $|S|$ labels (each node in S_i has $|S_i|$ labels) where $S_i \cap S_j = \emptyset$ for any $i \neq j$ and $|S_i| = \Theta(|S|/k) \geq \Omega(\log n)$. Our goal is to output k nodes with k representative labels. We use RepresentativeK(S) to solve this problem.

▶ **Lemma 12.** *The RepresentativeK(S) problem can be solved in $O(\log_s n)$ rounds where S is created by RandomSet and $|S| \geq \Omega(k \log n)$ in the s-space MPC model ($s = \Omega(\log n)$).*

CompareCut(S, V)

In the CompareCut(S, V) problem, the input is a set S of k sets, i.e., $\{S_1, S_2, \ldots, S_k\}$ and the vertex set V, the goal is to output the largest one among numbers $n(u, S_i)$ of edges between $u \in V$ and S_i for each S_i, along with the label of u (i.e., the label of the S_i), where $i \in [k]$. We use ComputeCut(S,V) to solve this problem.

▶ **Lemma 13.** *Given a graph $G = (V, E)$, let $S \subset V$ be a random set of nodes created by RandomSet. The CompareCut(S, V) problem can be solved in $O(\log_s n)$ rounds in the s-space MPC model ($s = \Omega(\log n)$).*

4 The Algorithm Based on Neighbor Counting

Recall that a graph $G = (V, E)$ is generated from the SBM(n, p, q, k) if there is a hidden partition $V = \cup_{i=1}^{k} V_k$ of the nk-vertex set V, and for any two vertices u, v that belong to the same cluster, the edge (u, v) appears in E with probability p; for any two vertices u, v that belong to two different clusters, the edge (u, v) appears in E with probability q, where $0 < q < p < 1$. In this section, we give the algorithm underlying Theorem 3.

We first give a simple sequential algorithm based on comparing common neighbors. Then we show how to implement it in the s-space model. To do so, we give implementations of a number of basic graph operations in the s-space model, which is deferred to Section 3.

4.1 A Sequential Algorithm Based on Counting Common Neighbors

▶ **Theorem 14.** *If* $\frac{p-q}{\sqrt{p}} \geq \Omega(\frac{(k+1)^{1/2}}{n^{1/4}})$*, the algorithm* CommNBR *can output* k *clusters in* $O(\frac{k^2 n \log n}{p})$ *time with probability* $1 - O(\frac{1}{n})$.

In our sequential algorithm CommNBR, we first randomly sample a set S of $\frac{21nk^2 \log n}{d}$ nodes from V such that each cluster has more than $\Theta(\log n)$ nodes with high probability where d is the number of neighbors of an arbitrary node $u \in V$. Then, for each pair u, v, we count the number of their common neighbors in G, i.e., those vertices that are connected to both u and v. If the number of common neighbors is above some threshold Δ, then we put them into the same cluster. In this way, we can obtain k *sub-clusters* of S, $C_1, \ldots, C_k$. That is, each $C_i \subseteq S$ and is a subset of some cluster, i.e., $C_i = V_{\pi(i)} \cap S$ for some permutation $\pi : \{1, \ldots, k\} \to \{1, \ldots, k\}$. Let $L(v, C_i)$ denote the number of incident edges between a node v and a cluster C_i. We can then cluster each remaining node $v \in V \setminus S$ by finding the index j such that $L(v, C_j)$ is the greatest among all numbers $L(v, C_i), 1 \leq i \leq k$.

For the intuition of the existence of such a threshold Δ, let us take the case $k = 2$ as an example. In this case, for any two vertices u, v belonging to the same cluster, the expected number of common neighbors is $p^2 n + q^2 n$; for any two vertices u, v belonging to two different clusters, the expected number of common neighbors is $2npq$. Since $\frac{p-q}{\sqrt{p}} \geq \Omega\left(\frac{(k+1)^{1/2}}{n^{1/4}}\right)$, there exists a sufficiently large gap between these two numbers so that we can define a suitable threshold. However, the values of p and q are not provided. To address this issue, we propose an algorithm called ComputeDEL, which can be described as follows. We first sample a set $\mathcal{S}_\Delta$ of $\Theta(k \log n)$ nodes, and for each pair $u, v \in \mathcal{S}_\Delta$, we compute a set $\mathcal{V}_\Delta$ of values of $|N(u) \cap N(v)|$. We let $\Delta' = \max\{value \in \mathcal{V}_\Delta\}$, and then we have $\Delta = \Delta' - 9\sqrt{\Delta' \log n}$.

4.2 Implementation in the s-space MPC model

Now we describe our MPC algorithm MPC-CommNBR, which is an implementation of CommNBR, where the local memory is $s = \Omega(\log n)$, and prove Theorem 3.

Recall that in CommNBR, there are two major steps. In the first step, we need to find k clusters from a set S of randomly sampled nodes. In the second step, based on the clustering on S, we cluster all nodes in V. The major challenge lies in the simulation (in MPC model) of the first step which is to compare common neighbors between two nodes u and v. It is easy to see that computing $|N(u) \cap N(v)|$ for $u, v \in V$ is exactly the task of finding common elements in two sets. For convenience, we use a set S_u to denote $N(u)$ for a node $u \in V$. Then, we need to find the common elements between S_u and S_v by a method $\textsc{comm}(\mathsf{S}_u, \mathsf{S}_v)$. Note that we need to execute $\textsc{comm}(\mathsf{S}_u, \mathsf{S}_v)$ for different u and v for many times. Therefore, to compute $|N(u) \cap N(v)|$ for different $u, v \in V$ efficiently, we need to solve two problems.

The first one is to implement $\textsc{comm}(\mathsf{S}_u, \mathsf{S}_v)$ efficiently in MPC model where each machine's memory is $s = \Omega(\log n)$. The second one is to execute the first one in parallel. For the first one, we use a simple method to implement $\textsc{comm}(\mathsf{S}_u, \mathsf{S}_v)$ in MPC model. Let V' be the set of nodes in which for each $u \in V'$, S_u will be compared. We make each byte of $\mathsf{S}_u (u \in V')$ aligned. Then, we can directly compute $|\mathsf{S}_u \cap \mathsf{S}_v|$. For the second one, we solve it by copying sets for enough times and then we let machines storing these copied sets execute the same algorithm in parallel. We use the MPC implementations of the basic graph operations in Section 3 to implement our clustering algorithm here.

Algorithm 1 MPC-CommNBR: An MPC algorithm based on counting common neighbors.

Input: A SBM graph G;

1: Let $d = |N(u)|$ where u is an arbitrary node in V
2: Apply RandomSet to obtain random node set S' and S, where $|S'| = \Theta(k \log n)$ and $|S| = (21k^2 n \log n)/d$
3: Update S' and obtain Δ by ComputeREP(S')
4: Obtain k sub-clusters $S_1, \ldots, S_k$ by ComputeSubcluster(S, S', Δ)
5: Obtain k clusters of V by ComputeCluster($S_1, \ldots, S_k, V$).

Algorithm 2 ComputeRep: Compute representative for each cluster by common neighbors and Δ.

Input: Random vertex set S';

1: Run CompareINIT(S');
2: Obtain k nodes with k representative lables by RepresentativeK(S');
3: Update S' by only keeping k nodes obtained from Step 2.
4: **return** S' and Δ

For the algorithm CompareINIT(S'), we describe it as follows.

CompareINIT(S'):
1) Reorganize neighbors of nodes in S' by ReorganizeNBR(S').
2) Execute CopyNBR(S') to create $|S'|$ copies of $N(u)$ for each $u \in S'$.
3) Based on copies of $N(u)$ from 2), we directly compute $|N(u) \cap N(v)|$ in parallel where $u, v \in S'$.
4) Execute CompareGRP to obtain final results by summing all partial results obtained from 3).
5) Compute Δ', i.e., the maximum value of $|N(u) \cap N(v)|$ for all pairs of $u, v \in S'$ and output $\Delta = \Delta' - 9\sqrt{\Delta' \log n}$.

In ComputeSubcluster(S', S, Δ), the process is similar to ComputeREP(S'). The major difference is that we only copy $N(u)$ for each $u \in S$ for k times and copy $N(v)$ for each $v \in S'$ for $|S|$ times. By computing $|N(u) \cap N(v)|$ where $u \in$ copy-S, $v \in$ copy-S', and copy-S, copy-S' are the copies of S and S', we can obtain k sub-clusters of S. The details of computing can refer to ComputeREP(S').

In ComputeCluster($S_1, \ldots, S_k, V$), we first apply EvenCluster(S) to output k sub-clusters from S such that each cluster has the same number of nodes. Then, we use ComputeCut(S, V) to cluster V.

Next, we show the details of CompareGRP used in the procedure of ComputeREP(S'). In the CompareGRP problem, the input is a set of groups of machines and the goal is to output the results by comparing groups of machines. Take two groups A, B of machines as

an example. Our goal is to output the common elements by comparing A and B. We say that group A compares to group B which means that the i-th member machine of the group A will compare to the i-th member machine of group B ($i \in [n/s]$).

▶ **Lemma 15.** *Given a set of consecutive groups each of which has n/s member machines, there exists an algorithm that takes $O(\log_s n)$ rounds to obtain the results of comparing data between groups correspondingly.*

Now we are ready to prove Theorem 3.

Proof of Theorem 3. The correctness of obtaining k clusters based on counting common neighbors can be seen in Theorem 14. By Lemma 8, we can create randomly sampled sets S and S' such that each machine M knows indexes of nodes in $M \cap S$ and $M \cap S'$ within $O(\log_s n)$ rounds.

Next, we first prove that by ComputeREP(S'), we can obtain k sub-clusters within $O(\log_s n)$ rounds. By Lemma 9, it takes $O(\log_s n)$ rounds for ReorganizeNBR(S'). By Lemma 10, we use $O(\log_s n)$ rounds to finish CopyNBR(S') for each $N(u)$ where $u \in S'$. By Lemma 15, it takes $O(\log_s n)$ to first get partial results and then obtain the complete results of $|N(u) \cap N(v)|$ where $u, v \in S'$. We can use $O(\log_s n)$ rounds to obtain Δ by simulating ComputeDEL within $O(\log_s n)$ rounds. Then by Lemma 15 and Lemma 12, we can obtain k sub-clusters from S' in $O(\log_s n)$ rounds in the MPC model and the total space is $\tilde{O}(m)$. Similarly, by ComputeSubcluster(S', S, Δ), we can prove that within $O(\log_s n)$ rounds, we can obtain k clusters from S in the MPC model and the total space used is $\tilde{O}(m)$.

Now, let us see the last step of obtaining k clusters of V. By Lemma 11 and setting $|S^*| = \frac{20k^2 n \log n}{d}$, we can output $S^* \subseteq S$ such that for any two labels $i, j \in [k]$, we have $\mathsf{N}_{S^*}(i) = \mathsf{N}_{S^*}(j)$ in $O(\log_s n)$ rounds, where $\mathsf{N}_{S^*}(i)$ is the number of nodes in S^* with label i. Finally, by Lemma 13, we can decide all labels of V within $O(\log_s n)$ rounds with high probability. The total space used in MPC model is $\tilde{O}(m)$. Thus, our proof is completed. ◀

5 The Algorithm Based on Power Iterations

In this section, we give another MPC algorithm for a general SBM graph in the s-space model and prove Theorem 1. The omitted proof of this section is deferred to the full version. Also, we first carefully design a sequential algorithm and then we implement it in the s-space MPC model. Our second sequential algorithm is based on power iterations which perform well in a recent algorithm in [25]. The algorithm makes use of the adjacency matrix A of the graph G, from which we define a matrix $B = A - q \cdot J$, where J is the $n \times n$ all-1 matrix. Then it decides if two vertices u, v are in the same cluster or not by checking the ℓ_2-norm of the difference between B_u^r and B_v^r, which are the rows corresponding to vertices u, v, respectively, in the matrix B^r (the r-th power of matrix B).

We note that the algorithm in [25] only considers the special case that $r = \log n$. Here, our sequential algorithm considers all possible $r \in \{1, \ldots, O(\log n)\}$ and for each r we choose a different threshold Δ, which depends on the value of p, q and k. To implement our sequential algorithm in the MPC model, we divide the process of matrix computation into different components each of which can be implemented efficiently in the s-space model.

5.1 A Sequential Algorithm Based on Power Iterations

We now describe Algorithm PowerIteration. Let A be an adjacency matrix of the input graph G and r where r is a parameter. Let $\Delta = C\sqrt{k}\sqrt{p(1-q)}(\log kn)^7(p-q)^{r-1}n^{r-1}$, where $C > 0$ is some universal constant. We set $B = A - q \cdot J$ where $J = 1^{n \times n}$. Let $W = V$.

We choose an arbitrary vertex v in W and put v into a sub-cluster C_i where $i \in [k]$. For each node u in W, if $\|B_u^r - B_v^r\| \leq \Delta$, then we add u to C_i. Next, we remove C_i from W. Repeat the above process until W is empty. Then we return all the clusters C_i's.

▶ **Theorem 16.** *Let $p, q \leq 0.75$ be parameters such that $\max\{p(1-p), q(1-q)\} \geq C_0 \log n/n$ where $C_0 > 0$ is some constant. Suppose that $\frac{p-q}{\sqrt{p}} \geq (C_0^2+1)k^{\frac{1}{2}} n^{-\frac{1}{2}+\frac{1}{2(r-1)}} (\log kn)^7$. Let G be a random graph generated from SBM(n, p, q, k) and $r \in [3, O(\log n)]$, then with probability at least $1 - O(n^{-1})$ the algorithm PowerIteration can output k hidden clusters for suitable Δ and r.*

5.2 The MPC Algorithm

In this section, we show how to implement PowerIteration in the s-space MPC model. The pseudocode is found in Algorithm 3. Given a matrix A, we use A_i^{2r} to denote the i-th row of A^{2r}. We use $A_{\cdot j}^{2r}$ to denote the j-th column of A^{2r}.

Algorithm 3 MPC-PowerIteration.

Input: A SBM graph G, Δ;

Let A be an adjacent matrix of G, r be the parameter
Let $\Delta = C\sqrt{k}\sqrt{p(1-q)}(\log kn)^7(p-q)^{r-1}n^{r-1}$, where $C \in \mathbb{R}_*^+$
$B = A - q \cdot J$ where $J = 1^{n\times n}$
Let $i = 1$ and $W = V$
Initially, all nodes in W are active
while IsActive(W) is true **do**
  Choose an arbitrary vertex $v \in W$ and send it to all machines
  Label v with i, i.e., $C_i = \{v\}$
  for each machine holding active vertex $u \in W$, we execute the following procedure in parallel **do**
    ComputeNorm(B, u, v, r)
    if $\|B_u^r - B_v^r\| \leq \Delta$ **then**
      Label u with i
  Set nodes in C_i inactive
  $i = i + 1$
Return all the sub-clusters C_i's.

We use IsActive(W) to determine whether there are active nodes in W or not, which can be done in $O(\log_s n)$ rounds. The details of IsActive(W) is given in the full version.

We then use another subroutine ComputeNorm(B, i, j, r) to compute $\|B_i^r - B_j^r\|$. Notice that we can't directly calculate matrix multiplication, which will take lots of rounds, we notice some good properties of $\|B_i^r - B_j^r\|$ and have the following theorem.

▶ **Theorem 17.** *For a fixed i and $j \in [n]$ and any integer $r < O(\log n)$, the subrountine ComputeNorm(B, i, j, r) for computing $\|B_i^r - B_j^r\|$ can be implemented in $O(r \log_s n)$ rounds where each machine has memory $s = \Omega(\log n)$.*

Now we give the ideas of ComputeNorm(B, i, j, r). We find that the expansion of $\|B_i^r - B_j^r\|$ has good properties such that we only need to compute the key terms for these $O(r^2)$ terms and the coefficients have good combinatorial explanations. Then we can use graph algorithms to calculate the results. First we note that

$$||B_i^r - B_j^r||^2 = ||(\mathbf{1}_i^T - \mathbf{1}_j^T)(A - qJ)^r||^2 = (\mathbf{1}_i^T - \mathbf{1}_j^T)(A - qJ)^{2r}(\mathbf{1}_i - \mathbf{1}_j)$$
$$= \mathbf{1}_i^T(A - qJ)^{2r}\mathbf{1}_i - \mathbf{1}_i^T(A - qJ)^{2r}\mathbf{1}_j - \mathbf{1}_j^T(A - qJ)^{2r}\mathbf{1}_i + \mathbf{1}_j^T(A - qJ)^{2r}\mathbf{1}_j,$$

so we only need to calculate $\mathbf{1}_x^T(A - qJ)^{2r}\mathbf{1}_y$ where $x, y \in \{i, j\}$.

Now we have the following lemma about the expanded formula.

▶ **Lemma 18.** *We have*

$$\mathbf{1}_x^T(A - qJ)^{2r}\mathbf{1}_y = \mathbf{1}_x^T A^{2r}\mathbf{1}_y + \sum_{0 \le i_1 \le 2r-1} \sum_{0 \le i_t \le 2r-1} X_{i_1,i_t}(A_x^{i_1})J(A_{\cdot y}^{i_t})$$

where X_{i_1,i_t} is coefficient only related to n, q and C_i and C_i is the total number of different walks with length i from n vertices.

Since $r = O(\log n)$, there are $\text{poly}(\log n)$ terms in the right hand side. So we can store all coefficients in $\tilde{O}(n)$ space. Notice that $\mathbf{1}_x^T A^{2r}\mathbf{1}_y$ for all $y \in [n]$ is the i^{th} row of A^{2r}, i.e., A_x^{2r}. To compute $\mathbf{1}_x^T(A - qJ)^{2r}\mathbf{1}_y$, we split it into computing C_i, $A_x^{i_1}JA_{\cdot y}^{i_t}$, and A_x^{2r}.

Compute C_i and $A_x^{i_1}JA_{\cdot y}^{i_t}$

We show how to compute the value of any C_i and $A_x^{i_1}JA_{\cdot y}^{i_t}$. Let $\vec{\mathbf{j}}$ be all ones vector, which is a column of J. To compute $A_x^i\vec{\mathbf{j}}$, we propose a simple algorithm, i.e., Algorithm 4 that is described as follows.

Algorithm 4 AixSum$(G(n), r)$: Calculating $A_x^i\vec{\mathbf{j}}$ for all $x \in [n], i \in [2r]$.

Input: A graph $G(n)$,r
1: **for** each node u in G **do**
2: $A_{u,0} = 1$
3: let $i = 1$
4: **while** $i \le 2r$ **do**
5: **for** each node u in G **do**
6: $sum_u = 0$
7: **for** each neighbor v of u **do**
8: $sum_u += A_{v,i-1}$
9: $A_{v,i} = sum_u$
10: $i = i + 1$
11: Return A, $A_{x,i}$ is $A_x^i\vec{\mathbf{j}}$

▶ **Lemma 19.** *For all $x \in [n], i \in [2r]$, Algorithm 4 outputs $A_x^i\vec{\mathbf{j}}$.*

To compute C_i for any $i \in [n]$, we only need to sum up $A_x^i\vec{\mathbf{j}}$ for all $x, i \in [n]$.

Now, let us see how to implement Algorithm 4 in the s-space MPC model. Notice that in default, we use the fact that in the s-space MPC model, each vertex can visit its neighbors in $O(\log_s n)$ rounds where each machine has memory of $O(n^\delta)$ by Lemma 5.

▶ **Lemma 20.** *In s-space MPC model, for all $i \in [2r]$ and $x \in [n]$, there exists an algorithm that can compute all $A_x^i\vec{\mathbf{j}}$ in $O(r \log_s n)$ rounds,where each machine has memory $s = \Omega(\log_s n)$.*

Notice that $A_x^{i_1}JA_{\cdot y}^{i_t} = A_x^{i_1}\vec{\mathbf{j}}(A_y^{i_t}\vec{\mathbf{j}})$, $C_i = \sum_{x\in[n]} A_x^i\vec{\mathbf{j}}$ and we have computed $A_x^i\vec{\mathbf{j}}$ for all $i \in [2r]$ and $x \in [n]$, we can obtain $A_x^{i_1}JA_{\cdot y}^{i_t}$ in constant rounds. So the main round complexity is only about the calculation of $A_x^i\vec{\mathbf{j}}$ and we have the following corollary.

▶ **Corollary 21.** *In s-space MPC model, there exists an algorithm that can compute $A_x^{i_1}JA_{\cdot y}^{i_t}$ and C_i in $O(r \log_s n)$ rounds, where each machine has memory $s = \Omega(\log n)$.*

Compute A_x^{2r}

Note that each entry $a_{x,y}^{2r}$ in A_x^{2r} is exactly the number of walks with step size r from v_x to v_y. The naive algorithm of computing A^{2r} is to compute the matrix, but it is resources-consuming. Another idea is to compute $a_{x,y}^{2r}$ for any x and y, respectively. If each machine can store all vertices within radius r, then we can directly compute all $a_{x,y}^{2r}$ $(x, y \in [n])$. Now, we consider the s-space MPC model, i.e., single machines cannot store vertices within radius r.

▶ **Lemma 22.** *Let $a_{x,y}^{2r}$ be the number of walks from the vertex x to the vertex y after walks with step size $2r$. There exists a procedure* ComputeArx *that can find $a_{x,y}^{2r}$ after $O(r \log_s n)$ rounds where each machine has memory $s = \Omega(\log n)$.*

By taking the union of different vertices, we can get the following corollary.

▶ **Corollary 23.** *Let A_i^{2r} be set of the numbers of walks from the vertex i to the vertex j where $j \in [1, n]$ after walks with step size r. The procedure* ComputeArx *can find A_i^{2r} after $O(r \log_s n)$ rounds where each machine has memory $s = \Omega(\log n)$.*

Compute $\|B_i^r - B_j^r\|$

After obtaining A_x^i and $A_{\cdot y}^i$, the value of $A_x^i J A_{\cdot y}^i$ is the multiplication of the sums of terms in each of two vectors. And the coefficient of each term is the multiplication of C_i, n and q. Now, we can prove Theorem 17.

Proof of Theorem 17. By Lemma 18, $\mathbf{1}_x^T (A - qJ)^{2r} \mathbf{1}_y$ which consists of at most $O(r^2)$ terms with C_i, $A_x^{i_1} J A_{\cdot y}^{i_t}$, and A_x^{2r}. Let us see the round complexity of computing it. We take the round complexity of computing one key term as an example. We only need to look at the round complexity of computing $\left(\prod_{l=2}^{t-1} C_{i_l}\right) (A_x^{i_1}) J (A_{\cdot y}^{i_t})$. By Corollary 21, we need $O(i \log_s n)$ rounds to compute any C_i where $i \in [2r]$. We can finish calculating $\left(\prod_{l=2}^{t-1} C_{i_l}\right)$ in $O(r \log_s n)$ rounds. By Lemma 20, we can obtain the result of $(A_x^{i_1}) J (A_{\cdot y}^{i_t})$ within $O(r \log_s n)$ rounds. Notice that there is a special term $\mathbf{1}_x^T A^{2r} \mathbf{1}_y^T$. By Corollary 23, we can find it within $O(r \log_s n)$ rounds. There are some other similar computations, which also take $O(r \log_s n)$ rounds. Recall that there are $O(r^2)$ terms, we deal with it by copying the whole graph for $\mathrm{poly}(\log n)$ times and then put these results together. Therefore, it takes $O(r \log_s n)$ rounds to calculate $\mathbf{1}_x^T (A - qJ)^{2r} \mathbf{1}_y$. Therefore, we need $O(r \log_s n)$ rounds to finish the calculation of $\|B_i^r - B_j^r\|$. ◀

By Theorem 17, we can finish the proof of Theorem 1.

Proof of Theorem 1. The main idea of MPC-PowerIteration(Algorithm 3) is to fix a node v_i first and calculate $\|B_i^r - B_j^r\|$ for any other node v_j in the same cluster to obtain all nodes in the same cluster. We need to store all simplified coefficients in each round which uses $O(nr^2)$ space. For other operations in the algorithms, $O(m)$ space is enough. So the total space complexity is $\tilde{O}(m + nr^2) = \tilde{O}(m)$.

In the full version, we show how to implement IsActive(W) in $O(\log_s n)$ rounds. By Theorem 17, we can finish $\|B_i^r - B_j^r\|$ within $O(r \log_s n)$ rounds. Therefore, we need $O(r \log_s n)$ rounds to find a cluster and all its nodes. There are k hidden clusters and we execute the above procedure sequentially, so the round complexity is $O(kr \log_s n)$. ◀

Now we show how to use more space to trade off round complexity and give the proof of Theorem 2.

Proof of Theorem 2. Recall that in MPC-PowerIteration(Algorithm 3), we sequentially find k clusters, that is the reason why there is a factor k in the round complexity. Now, we execute the process in parallel. First, we randomly sample a set S_k of $\Theta(k \log n)$ nodes. With high probability, for each hidden cluster, we sample $\Theta(\log n)$ nodes in S_k. Then, for each node $u \in S_k$, we execute ComputeMatrix(B, u, v, r) for each $v \in V$. If $\|B_u^r - B_v^r\| \leq \Delta$, we put u and v in the same cluster. The space for this step is $\tilde{O}(km + knr^2 \log n)$. Then, we will have k clusters with $\Theta(k \log n)$ labels. We remove duplicated labels by keeping the label with the minimum value among all received labels to get one label vertex for each cluster. Then by using these k label vertices, we can use $\tilde{O}(km + knr^2 \log n) = \tilde{O}(km)$ space to cluster all vertices. So, we can find all k clusters in $O(r \log_s n)$ rounds with high probability. ◀

References

1. Emmanuel Abbe. Community detection and stochastic block models. *Found. Trends Commun. Inf. Theory*, 14(1-2):1–162, 2018.
2. Emmanuel Abbe, Afonso S. Bandeira, and Georgina Hall. Exact recovery in the stochastic block model. *IEEE Trans. Inf. Theory*, 62(1):471–487, 2016.
3. Sepehr Assadi and Chen Wang. Sublinear time and space algorithms for correlation clustering via sparse-dense decompositions. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 – February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 10:1–10:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
4. MohammadHossein Bateni, Hossein Esfandiari, Manuela Fischer, and Vahab S. Mirrokni. Extreme k-center clustering. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 3941–3949. AAAI Press, 2021.
5. Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In Richard Hull and Wenfei Fan, editors, *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA – June 22 – 27, 2013*, pages 273–284. ACM, 2013.
6. Soheil Behnezhad, Sebastian Brandt, Mahsa Derakhshan, Manuela Fischer, MohammadTaghi Hajiaghayi, Richard M Karp, and Jara Uitto. Massively parallel computation of matching and mis in sparse graphs. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 481–490, 2019.
7. Soheil Behnezhad, Moses Charikar, Weiyun Ma, and Li-Yang Tan. Almost 3-approximate correlation clustering in constant rounds. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 – November 3, 2022*, pages 720–731. IEEE, 2022.
8. Aditya Bhaskara and Maheshakya Wijewardena. Distributed clustering via LSH based data partitioning. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 569–578. PMLR, 2018.
9. Guy E. Blelloch, Jeremy T. Fineman, and Julian Shun. Greedy sequential maximal independent set and matching are parallel on average. In Guy E. Blelloch and Maurice Herlihy, editors, *24th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '12, Pittsburgh, PA, USA, June 25-27, 2012*, pages 308–317. ACM, 2012.
10. Mélanie Cambus, Davin Choo, Havu Miikonen, and Jara Uitto. Massively parallel correlation clustering in bounded arboricity graphs. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)*, volume 209 of *LIPIcs*, pages 15:1–15:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

11 Flavio Chierichetti, Nilesh N. Dalvi, and Ravi Kumar. Correlation clustering in MapReduce. In *Proceedings of the 20th International Conference on Knowledge Discovery and Data Mining*, pages 641–650. ACM, 2014.

12 Vincent Cohen-Addad, Adrian Kosowski, Frederik Mallmann-Trenn, and David Saulpic. On the power of louvain in the stochastic block model. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

13 Vincent Cohen-Addad, Silvio Lattanzi, Slobodan Mitrovic, Ashkan Norouzi-Fard, Nikos Parotsidis, and Jakub Tarnawski. Correlation clustering in constant many parallel rounds. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 2069–2078. PMLR, 2021.

14 Vincent Cohen-Addad, Silvio Lattanzi, Ashkan Norouzi-Fard, Christian Sohler, and Ola Svensson. Parallel and efficient hierarchical k-median clustering. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 20333–20345, 2021.

15 Vincent Cohen-Addad, Frederik Mallmann-Trenn, and David Saulpic. Community recovery in the degree-heterogeneous stochastic block model. In Po-Ling Loh and Maxim Raginsky, editors, *Conference on Learning Theory, 2-5 July 2022, London, UK*, volume 178 of *Proceedings of Machine Learning Research*, pages 1662–1692. PMLR, 2022.

16 Vincent Cohen-Addad, Frederik Mallmann-Trenn, and David Saulpic. A massively parallel modularity-maximizing algorithm with provable guarantees. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 – 29, 2022*, pages 356–365. ACM, 2022.

17 Vincent Cohen-Addad, Vahab S. Mirrokni, and Peilin Zhong. Massively parallel k-means clustering for perturbation resilient instances. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 4180–4201. PMLR, 2022.

18 Sam Coy, Artur Czumaj, and Gopinath Mishra. On parallel k-center clustering. *arXiv preprint arXiv:2304.05883*, 2023.

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

20 Alessandro Epasto, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Scalable diversity maximization via small-size composable core-sets (brief announcement). In Christian Scheideler and Petra Berenbrink, editors, *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 41–42. ACM, 2019.

21 Manuela Fischer and Andreas Noever. Tight analysis of parallel randomized greedy MIS. *ACM Trans. Algorithms*, 16(1):6:1–6:13, 2020.

22 Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In Takao Asano, Shin-Ichi Nakano, Yoshio Okamoto, and Osamu Watanabe, editors, *Algorithms and Computation – 22nd International Symposium, ISAAC 2011, Yokohama, Japan, December 5-8, 2011. Proceedings*, volume 7074 of *Lecture Notes in Computer Science*, pages 374–383. Springer, 2011.

23 Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In Paulo Ferreira, Thomas R. Gross, and Luís Veiga, editors, *Proceedings of the 2007 EuroSys Conference, Lisbon, Portugal, March 21-23, 2007*, pages 59–72. ACM, 2007.

24 Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 938–948. SIAM, 2010.

25 Chandra Sekhar Mukherjee and Jiapeng Zhang. Detecting hidden communities by power iterations with connections to vanilla spectral algorithms, 2022. doi:10.48550/ARXIV.2211.03939.

26 Xinghao Pan, Dimitris S. Papailiopoulos, Samet Oymak, Benjamin Recht, Kannan Ramchandran, and Michael I. Jordan. Parallel correlation clustering on big graphs. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 82–90, 2015.

27 Peng Wang, Zirui Zhou, and Anthony Man-Cho So. A nearly-linear time algorithm for exact community recovery in stochastic block model. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 10126–10135. PMLR, 2020.

28 Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.

29 Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In Erich M. Nahum and Dongyan Xu, editors, *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, Boston, MA, USA, June 22, 2010*. USENIX Association, 2010.