Fully-Scalable Massively Parallel Algorithm for k-center with Outliers

Di Wu [1,2], Qilong Feng [1,2,*], Junyu Huang [1,2], Jinhui Xu [3], Ziyun Huang [4], Jianxin Wang [1,2,5,*]

[1] School of Computer Science and Engineering, Central South University, Changsha 410083, China
[2] Xiangjiang Laboratory, Changsha 410205, China
[3] Department of Computer Science and Engineering, State University of New York at Buffalo, NY, USA
[4] Department of Computer Science and Software Engineering, Penn State Erie, The Behrend College
[5] The Hunan Provincial Key Lab of Bioinformatics, Central South University, Changsha 410083, China
csuwudi@csu.edu.cn, csufeng@mail.csu.edu.cn, junyuhuangcsu@foxmail.com, jinhui@buffalo.edu, zxh201@psu.edu, jxwang@mail.csu.edu.cn

Abstract

In this paper, we consider the k-center problem with outliers (the (k, z)-center problem) in the context of Massively Parallel Computation (MPC). Existing MPC algorithms for the (k, z)-center problem typically require $\Omega(k)$ local space per machine. While this may be feasible when k is small, these algorithms become impractical for large k, where each machine may lack sufficient space for computation. This motivates the study of fully-scalable algorithms with sublinear local space. We propose the first fully-scalable MPC algorithm for the (k, z)-center problem. The main challenge is to design an MPC algorithm that operates with sublinear local space for finding the inliers close to the optimal clustering centers, and ensuring the approximation loss remains bounded. To address this issue, we propose an iterative sampling-based algorithm with sublinear local space in the data size. A key component of our approach is an outliers-removal algorithm that adjusts the sample size in each iteration to select inliers as clustering centers. However, the number of discarded inliers increases with the iteration of the outliers-removal algorithm, making it difficult to bound. To address this, we propose a self-adaptive method that can automatically adjust sample size to account for different data distributions on each machine, ensuring a lower bound on the sampling success probability. With these techniques, we present an $O(\log^* n)$-approximation MPC algorithm for the (k, z)-center problem in constant-dimensional Euclidean space. The algorithm discards at most $(1 + \epsilon)z$ outliers, completing in $O(\log \log n)$ computation rounds while using $\Theta(n^\delta)$ local space per machine.

Introduction

Clustering is one of the most fundamental computational tasks in the field of machine learning. The goal of clustering is to partition the given dataset into several clusters such that the points in the same cluster are more similar to each other, while the points in different clusters have less similarity. Among different clustering objectives, the k-center is an important clustering formulation, where the goal is to minimize the maximum distances between data points to their closest clustering centers. The k-center problem has been widely studied over the past decades and was also known to be NP-hard even in a plane (Megiddo and Supowit 1984; Feder and Greene 1988). Gonzalez (Gonzalez 1985) demonstrated that a 2-approximate solution can be achieved with linear running time in the data size using a greedy strategy, matching the approximation lower bound of the problem.

In recent years, much attention has been paid to distributed and parallel computing (Ding et al. 2016; Guha, Li, and Zhang 2017; Chen, Azer, and Zhang 2018; Bateni et al. 2021; Cohen-Addad, Mirrokni, and Zhong 2022; Im et al. 2023; Coy, Czumaj, and Mishra 2023). MPC model is a theoretical model of parallel computation, capturing the major facets of large-scale computational systems such as MapReduce (Dean and Ghemawat 2008), Hadoop (White 2015), Dryad (Isard et al. 2007) and Spark (Zaharia et al. 2010). In the MPC model, the computation proceeds in rounds and each machine performs local computations. The goal is to design parallel algorithms that use machines with sublinear space and operate in a sublinear (and hopefully sublogarithmic) number of computation rounds. In parallel settings, the data often exhibit non-homogeneity and heterogeneity, making the MPC model sensitive to outliers. This sensitivity motivates us to study the clustering problem with outliers in the MPC model. The k-center problem is well-known for its sensitivity to outliers, as noted by (Charikar et al. 2001). One of the major challenges for clustering is how to deal with data noises. Charikar et al. (Charikar et al. 2001) formulated and studied the (k, z)-center problem. The (k, z)-center problem is to find a set of no more than k centers and a set of z outliers such that the maximum distance from all but at most z outliers of the input dataset to their nearest center is minimized. This variant has been widely studied in sequential algorithms (Ding, Yu, and Wang 2019; Chakrabarty, Goyal, and Krishnaswamy 2016; Ding et al. 2016; Guha, Li, and Zhang 2017; Chen, Azer, and Zhang 2018).

In this paper, we focus on the (k, z)-center problem in the MPC model, called MPC (k, z)-center problem. Malkomes et al. (Malkomes et al. 2015) proposed a 2-round $(13 + \epsilon)$-approximation MPC algorithm for the (k, z)-center problem with $\widetilde{O}(\sqrt{n(k + z)})$ space per machine. Ceccarello et al. (Ceccarello, Pietracaprina, and Pucci 2019) improved this result to a 2-round $(3 + \epsilon)$-approximation MPC algorithm

*Corresponding Authors

using $O(\sqrt{n(k+z)}(\frac{24}{\epsilon})^D)$ local space per machine, where D is the doubling dimension of doubling metrics. de Berg et al. (de Berg, Biabani, and Monemizadeh 2023) improved the above results to a 2-round $(1+\epsilon)$-approximation MPC algorithm for the (k,z)-center problem with $(\sqrt{nk/\epsilon^D} + \sqrt{n}\cdot\log(z+1)+z)$ space per machine. These MPC results use coresets with local space dependent on k and z, limiting their suitability for large k and z. Hence, as an important concept in the MPC model, the fully-scalable MPC model has been widely studied in recent years (Bateni et al. 2021; Cohen-Addad, Mirrokni, and Zhong 2022; Cao, Huang, and Su 2024). This model operates with sublinear local space in the data size, addressing the space limitations of traditional approaches. In the fully-scalable MPC model, a dataset of n points is distributed across m machines, each with sublinear local space $s = \Theta(n^\delta)$, where $\delta \in (0,1)$ is an arbitrarily small constant.

Coreset is a well-established data reduction technique that has been extensively studied for k-clustering problems in various settings, including distributed settings and parallel settings (Cohen-Addad, Saulpic, and Schwiegelshohn 2021; Braverman et al. 2022; Cohen-Addad et al. 2022; Bachem, Lucic, and Krause 2018). However, this method suffers from the drawback that each coreset must be stored in its entirety on a single machine, and all known coresets techniques require $\Omega(k)$ space (Cohen-Addad, Saulpic, and Schwiegelshohn 2021; Braverman et al. 2022; Cohen-Addad et al. 2022). Lower bounds indicate that the size of a coreset for k-clustering problems must be at least k (Cohen-Addad et al. 2022). Furthermore, many approaches for the (k,z)-center problem, such as coresets in (Malkomes et al. 2015; Ceccarello, Pietracaprina, and Pucci 2019; de Berg, Biabani, and Monemizadeh 2023) typically fail to yield fully-scalable MPC algorithms because these methods require local space of $\Omega(k)$ when implemented in the MPC model. Intuitively, this arises because a machine must track all k widely separated data points to avoid missing information about any cluster. Indeed, the k-clustering problem has many applications in scenarios involving a large value for k. A typical example of such a data processing scenario is federated learning (Karimireddy et al. 2020), which has been widely used in various machine learning tasks. In applications like detecting duplicates or compression (Cohen-Addad, Mirrokni, and Zhong 2022), k can reach hundreds of millions which is much larger than s, making it impractical to compute a solution of size $\Omega(k)$ on a single machine. As demonstrated in (Coy, Czumaj, and Mishra 2023), this makes it challenging to design a fully-scalable MPC algorithm for the (k,z)-center problem in the case of large k, especially when $k = \Omega((\log n)^c)$ for a constant $c \geq 1$.

In the fully-scalable MPC model, there are established results for standard k-clustering problems, such as k-center, k-means and k-median problems (Bateni et al. 2021; Cohen-Addad, Mirrokni, and Zhong 2022). For the k-means and k-median objectives, Cohen-Addad et al. (Cohen-Addad, Mirrokni, and Zhong 2022) introduced a $(1+\epsilon)$-approximate fully-scalable MPC algorithm in $O(1)$ rounds under the assumption of α-perturbation resilient instance. This algorithm utilizes a hierarchical clustering tree and a dynamic programming process over the tree. However, the existence of outliers complicates the construction of the hierarchical clustering tree, making it difficult to bound the approximation loss. For the k-center objective, Bateni et al. (Bateni et al. 2021) introduced an $O(\log\log\log n)$-approximation fully-scalable MPC algorithm that opens $k(1+o(1))$ centers in a sublogarithmic number of rounds. Coy et al. (Coy, Czumaj, and Mishra 2023) further improved this result by achieving an $O(\log^* n)$-approximation fully-scalable MPC algorithm. However, both algorithms in (Bateni et al. 2021) and (Coy, Czumaj, and Mishra 2023) are sensitive to outliers, making it challenging to accurately identify inliers that approximate optimal centers and to analyze the approximation ratio and number of rounds. Thus, developing a fully-scalable algorithm with an approximation guarantee in the presence of outliers remains a challenging task.

Our Contribution

In this paper, we propose a sampling-based fully-scalable MPC algorithm for the (k,z)-center problem in constant-dimensional Euclidean space, a setting commonly used in (Malkomes et al. 2015; Coy, Czumaj, and Mishra 2023; Bateni et al. 2021). Our proposed sampling-based approach addresses the challenge of scalability in local space and eliminates the dependence on the number of clusters k.

While many sampling methods have been proposed to solve the (k,z)-center problem (Charikar et al. 2001; Ding, Yu, and Wang 2019; Chakrabarty, Goyal, and Krishnaswamy 2016), designing sampling methods that operate with sublinear local space in the data size remains a key challenge in achieving a fully-scalable MPC algorithm. To address this challenge, we propose an outliers-removal sampling method designed to remove outliers while maintaining most inliers. The intuitive idea is to adjust the sample size in each machine to select inliers as clustering centers with a certain probability, such that most inliers can be covered. This approach ensures the approximation loss by the random ball covering process and guarantees that the number of discarded inliers remains within a bound of ϵz, where $\epsilon > 0$ is a parameter and z is the number of outliers. However, the outlier-removal sampling may incorrectly select outliers as centers during the random ball-cover process. To ensure coverage of most inliers, we might sample more than k points to cover optimal clusters, such that the number of opened centers on each machine is approximately bounded by $\Theta(n^\delta)$, leading to $\Theta(n^\delta \cdot m)$ opened centers across m machines. Then, we iteratively apply the outlier-removal sampling over $O(\log\log n)$ rounds to reduce the number of opened centers. However, the number of discarded inliers increases with the iteration rounds. To address this, we propose an (ϵ,η)-self adaptive method, where ϵ is a parameter about the number of discarded inliers and η is a parameter about success probability. This method can bound the number of discarded inliers and automatically adjust these key parameters to suit different data distributions on each machine. In each iteration, some "heavy" points with many assigned inliers might be discarded, causing the inliers assigned to them to be discarded as well. When outliers and inliers are closely mixed, complicating the tracking of inliers discarded near outliers.

Hence, we assume that each outlier is at least $O(4c_\rho r^*)$ away from any inlier, where r^* is the optimal radius. This assumption is reasonable in practical datasets, where outliers are typically far from inliers (Amagata 2024; Delic, Grcic, and Segvic 2024; Kanjanawattana 2019). Based on this assumption, we propose a τ-weight truncation method to control the number of outliers selected as centers and limit the number of discarded points. In each iteration, the weight of an opened center increases based on the total points assigned to it. If any uncovered inliers have weights exceeding τ, they are deemed "heavy" and cannot be discarded. This method utilizes weights to measure the importance of each point, ensuring that both the number of opened outliers and the number of discarded inliers can be bounded throughout the iteration process. With these techniques, we can obtain a fully-scalable MPC algorithm with $(1+\epsilon)z$ discarded outliers for the (k,z)-center problem. The main contributions of this paper are summarized as follows.

- We propose an outliers-removal sampling approach that guarantees approximation loss by random ball covering technique and ensures discarded inliers are bounded by ϵz. This method can adjust the sample size in each iteration to cover most inliers with a certain probability.
- We propose an (ϵ,η)-self adaptive method that automatically adjusts the sample size based on the key parameters to suit different data distributions in each iteration. This method ensures that the number of discarded inliers is bounded by ϵz throughout the iterations.
- We propose a τ-weight truncation method that uses weights to measure the importance of points. This method guarantees both the number of outliers selected as centers and the number of discarded inliers within bounds during the iterations.

We present our result in Theorem 1, where $\delta,\epsilon\in(0,1)$, and Δ denotes the ratio between the maximum and minimum pairwise distances in P. When under the assumption that Δ is bounded by $\mathrm{poly}(n)$, our MPC algorithm can operate with local space $s=\Theta(n^\delta)$ and total space $\widetilde{O}(n^{1+\delta})$.

Theorem 1. *Given a set $P\subset\mathbb{R}^d$ of n points, with constant probability, there exists a fully-scalable MPC algorithm that outputs an $O(\log^* n)$-approximate solution with at most $(1+\epsilon)z$ discarded outliers for the (k,z)-center problem using $k(1+o(1))+\frac{k\widetilde{O}(\log n)}{\epsilon}$ centers in $O(\log\log n)$ rounds. The MPC model requires local space $s=\Theta(n^\delta)$ and total space $\widetilde{O}(n^{1+\delta}\cdot\log^2\Delta)$.*

Preliminaries

Let P be the given set of data points with size n in d-dimensional Euclidean space, where d is a constant. For any two points $p,q\in\mathbb{R}^d$, let $d(p,q)$ denote their Euclidean distance. For a point $p\in P$ and a subset $S\subseteq P$, let $d(p,S)=\min_{s\in S}d(p,s)$. For any two subsets $S,S'\subseteq P$, let $d(S,S')=\min_{s\in S}d(s,S')$. Let $\nu(S,S')$ denote the maximum distance between the sets S and S' such that $\nu(S,S')=\max_{s\in S}d(s,S')$. Let Δ be the aspect ratio of P, where $\Delta=\frac{\max_{p,q\in P}d(p,q)}{\min_{p,q\in P,p\neq q}d(p,q)}$. For a positive integer l, let $[l]:=\{1,2,\ldots,l\}$. For a subset $S\subseteq P$, let $|S|$ denote the number of points in S. The notation $\log^* n$ represents the number of times the logarithm function must be applied before the result is less than or equal to 1. Formally, it is defined as follows: 1) $\log^* n=0$ if $n\leq 1$. 2) $\log^* n=1+\log^*(\log n)$ if $n>1$. This function grows extremely slowly. For example, $\log^* 2\approx 1$, $\log^* 16\approx 2$, and even for extremely large numbers like 10^{19}, $\log^* 10^{19}$ is still only about 5. Let $\log^{(i)} n:=\underbrace{\log\ldots\log}_{i} n$ be the iterated logarithm of n. Note that $\log^{(0)} n:=n$. For the given instance, we denote block B_h as a subset of points, where h is the center of B_h. For two hubs h and h' ($h\neq h'$), $B_h\cap B_{h'}=\emptyset$. Let c_ρ denote an LSH parameter, where $c_\rho\geq 1$ ($\rho\in(0,1)$) is a constant that depends only on ρ and is inversely proportional to it. Given a set $P\subset\mathbb{R}^d$ of n points, two integers $k\geq 1$ and $z\in[0,n]$, the goal of the k-center with outliers problem, called (k,z)-center problem, is to find a set C of k centers and a set $P'\subseteq P$ of $n-z$ points, such that the clustering cost $\nu(P',C)=\max_{p\in P'}d(p,C)$ is minimized. A clustering $\mathcal{C}$ of P is a partition of P' into nonempty clusters $\mathcal{C}_1,\ldots,\mathcal{C}_t$ ($t\in[k]$). Given a set $C\subseteq P$ of k centers, the optimal set P' can be derived from P by removing the z points $p\in P$ with the largest $d(p,C)$. Let a set C of k centers denote a solution to a (k,z)-center instance. We denote an optimal set of centers by C^* and the optimal radius by r^*. Based on C^*, let Z^* be the set of the furthest z outliers from C^*. Let $P_{opt}=P\backslash Z^*$ be the set of inliers. Based on C^*, by discarding the points in Z^*, we can find k optimal clusters, denoted by $D^*=\{D_1^*,\ldots,D_k^*\}$.

Computational Model. We consider the (k,z)-center problem under the MPC setting described in (Bateni et al. 2021; Coy, Czumaj, and Mishra 2023). The input dataset consists of n points distributed across m machines, each with a local space of $s=\Theta(n^\delta)$, where $\delta\in(0,1)$ is a constant. The MPC model has total space $\widetilde{O}(n^{1+\delta})$. Certain operations, including sorting, computing the prefix sum of n elements, and broadcasting a value smaller than s, can be computed deterministically in $O(1)$ rounds (Goodrich, Sitchinava, and Zhang 2011).

Remark 2. We assume that $z\geq\frac{k\widetilde{O}(\log n)}{\epsilon}$ ($\epsilon\in(0,1)$) for the outliers distribution, which is consistent with scenarios involving heavy noise (Li and Guo 2018; Jiang et al. 2019). Additionally, we assume that the distance between each outlier and inlier is larger than $\widetilde{O}(4c_\rho r^*)$, as outliers are typically far from inliers in practical datasets (Amagata 2024; Delic, Grcic, and Segvic 2024; Kanjanawattana 2019).

Algorithm for MPC (k,z)-Center

In this section, we present an MPC algorithm for the (k,z)-center problem, designed to use sublinear local space in the data size. The proposed approach is built on several critical components: outliers-removal sampling, (ϵ,η)-adaptive method and τ-weight truncation method. The high-level idea behind outliers-removal sampling is to distribute the dataset across m machines based on relative distances between data points and adjust the sample size in each machine to en-

sure that inliers are selected as clustering centers with a certain probability. This approach guarantees the approximation loss during the random ball covering process and ensures that the discarded inliers can be bounded by ϵz. Although the outliers are distant from the inliers, they might be densely clustered, the outliers-removal sampling might mistakenly select them as centers. To cover most inliers, we sample more than k points to cover most optimal clusters during the sampling process. Thus, the number of centers opened on each machine might be approximately bounded by $\Theta(n^\delta)$, potentially resulting in a total of $\Theta(n^\delta \cdot m)$ opened centers across m machines. To reduce the number of opened centers, we iteratively perform outliers-removal sampling over $O(\log\log n)$ rounds. This process reduces the number of opened centers in all the optimal clusters to $k(1+o(1))$ across all machines with an additional bounded approximation loss. However, the number of discarded inliers increases with the iteration. To bound the number of discarded inliers during the iteration process, we propose an (ϵ, η)-adaptive method, where ϵ is a parameter that controls the number of discarded inliers and η is a parameter that influences the success probability. This method can automatically adjust the sample size in each round to adapt to different data sizes, ensuring that the number of discarded inliers remains bounded. In the sampling-based approach, outliers might be mistakenly selected as centers, making it difficult to bound the number of outliers opened as centers. Meanwhile, some "heavy" points with many assigned inliers might be discarded, leading to assigned inliers also being discarded. When outliers and inliers are closely intermixed, this complicates estimates of discarded inliers near the outliers. Hence, we assume that each outlier is at least $O(4c_\rho r^*)$ away from any inlier. Under this assumption, we propose a τ-weight truncation method to bound the number of outliers opened as centers and control the number of discarded points during the iteration. The weight of an opened center after each iteration equals the total number of points assigned to it. If any uncovered inliers have weights exceeding τ, they cannot be discarded. This method can control the value of τ, ensuring that both the number of outliers selected as centers and the number of discarded points remain bounded over the iteration process. Due to the space limit, all the proofs are given in the full version.

Outliers-Removal Sampling Algorithm

In this section, we present an $O(1)$-approximation algorithm with at most ϵz inliers discarded. The outliers-removal sampling (called ORS) algorithm is described in Algorithm 1, where an intuitive idea is to use a two-stage strategy to sample points to cover most inliers. In the first stage (steps 1-5 of Algorithm 1), a block set is constructed to partition the dataset based on the relative distances between the pairwise points, breaking the original ties arbitrarily. This enables sparsification in the distances between any pair of points and ensures that points close to each other are grouped in a block together. Then, in the second stage (steps 6-20 of Algorithm 1), a random ball covering strategy is used in each block, ensuring a high probability of selecting most inliers. For Algorithm 1, the input is a subset Q of P. In Algorithm 1, the first sampling stage is to construct a block set to reorganize the dataset according to the relative distances between the data points, as outlined in steps 1-5. Given a probability p, each point in the dataset is sampled with probability p, forming a set of hubs $H \subseteq Q$. Then, we assign each point $q \in Q$ to its approximate nearest hub, and each hub h along with the points assigned to it form a block B_h. This process can be obtained by using the classical technique of locality-sensitive hashing (LSH). We start by introducing the definition of the LSH technique.

Definition 3. [LSH(Har-Peled, Indyk, and Motwani 2012)] Given $r \in \mathbb{R}^+$, $c > 1$ and $p_1, p_2 \in (0,1)$ where $p_1 > p_2$, a hash family $\mathcal{H} = \{h : \mathbb{R}^d \to U\}$ is a (r, cr, p_1, p_2)-LSH family, if $\forall x, y \in \mathbb{R}^d$, the following holds:

- If $d(x,y) \le r$, $\Pr_{h\in\mathcal{H}}[h(x) = h(y)] \ge p_1$;
- If $d(x,y) \ge cr$, $\Pr_{h\in\mathcal{H}}[h(x) = h(y)] \le p_2$.

As shown in (Har-Peled, Indyk, and Motwani 2012), we construct a $(r, c_\rho r, (1/n)^\rho, 1/n)$-LSH family.

Lemma 4. *(Har-Peled, Indyk, and Motwani 2012) Let $r, n \in \mathbb{N}$, and $\rho \in (0,1)$. There exists a $(r, c_\rho r, (1/n)^\rho, 1/n)$-LSH family, where c_ρ is a constant that depends only on ρ and is inversely proportional to it.*

In the following, we show how to use LSH to search for the approximate nearest hub in H for each point. Given a set Q of at most n points, where $H \subseteq Q$ is the set of hubs, our objective is to assign each point to its approximate nearest hub. The intuitive idea is to estimate the pairwise distance between each point and its nearest hub $\log\Delta$ times, and with each estimate, attempt to identify a hub within that approximate distance. For our $\log\Delta$ estimates for the correct r (the distance from each point to its nearest hub), we randomly select $\ell = \Theta(n^\rho)$ hash functions from a $(r, c_\rho r, (1/n)^\rho, 1/n)$-LSH family and use them to hash all the points independently and uniformly. Without loss of generality, we assume that $\rho < \delta$ as otherwise we can set $\rho = \delta$ (Coy, Czumaj, and Mishra 2023). Then, group all points with the same hash value on consecutive machines. Our purpose is to find a close hub with the same hash value for each point. Given a hash function f, if α hubs map to the same hash value and β points also map to this value, it becomes challenging when a large number of hubs share the same hash value. To address this issue, when many hubs are mapped to the same hash value, we retain only a constant number of hubs per hash value while ensuring that each point has a hub within a constant factor of the distance to the nearest hub. To increase the success probability, the approximate nearest hub search using LSH should be performed $O(\log n)$ times. The following lemma shows the approximation guarantee, number of rounds and space complexity for approximate nearest hub searching in MPC setting.

Lemma 5. *Given a set Q of at most n points in $\mathbb{R}^d$, the set $H \subseteq Q$ of hubs, and a constant c_ρ depending on ρ (where $\rho \in (0,1)$ is a constant), with probability at least $1 - \frac{1}{n^{\Omega(1)}}$, we can use LSH to find a hub $\mathrm{s}(q) \in H$ for all $q \in Q$ in $O(1)$ rounds such that $d(q, \mathrm{s}(q)) < 2c_\rho \cdot d(q, H)$. The nearest neighbor searching method uses local space $s = \Theta(n^\delta)$ and total space $\widetilde{O}(n^{1+\delta} \cdot \log\Delta)$.*

Algorithm 1: ORS $(Q, \omega, p, r^*, m, s, \epsilon, \eta, z, \rho, \tau)$

Input: A set $Q \subset \mathbb{R}^d$ of at most n points among m machines with local space s, probability parameter $p \in (0,1)$, outliers parameter $z \in [0, n]$, radius parameter $r^* \in \mathbb{R}^+$, integer m, parameter $s = \Theta(n^\delta)$ ($\delta \in (0,1)$), parameters $\epsilon, \eta, \rho \in (0,1)$, weight parameter $\tau \in \mathbb{R}^+$, a weighted function ω.
Output: A set $S \subseteq Q$ with a weighted function $\omega_S : S \to \mathbb{R}^+$ or "FAIL".

1: Sample each point in Q independently with probability p. Points which are sampled form the set of hubs H;
2: **if** $H = \emptyset$ **then**
3: Return "FAIL";
4: **end if**
5: For each point $q \in Q$, assign it to its approximate nearest hub in H by LSH, and let B_h be the set of points assigned to a hub $h \in H$;
6: **for** each $h \in H$ **do**
7: **if** $|B_h| \leq s$ **then**
8: Collect B_h on a single machine;
9: $(S_h, \omega_h) \leftarrow$ RS $(B_h, h, r^*, m, \epsilon, \eta, \omega, \tau)$;
10: **else**
11: Collect the points of B_h on consecutive machines to form sub-blocks $B_{h_1}, \ldots, B_{h_\lambda}$;
12: Put h in every B_{h_i} $(i \in [\lambda], \lambda \leq m)$;
13: Put other points in $B_h \backslash \{h\}$ exactly into every one of the sets B_{h_i}, such that $|B_{h_i}| \leq s = \Theta(n^\delta)$ for each $i \in [\lambda]$;
14: **for** each B_{h_i} $(i \in [\lambda])$ **do**
15: $(S_{h_i}, \omega_{h_i}) \leftarrow$ RS $(B_{h_i}, h, r^*, m, \epsilon, \eta, \omega, \tau)$;
16: **end for**
17: $S_h \leftarrow \bigcup_{i=1}^{\lambda} S_{h_i}$, and let ω_h be a weight function defined on S_h, such that for each point $s \in S_h$, $\omega_h(s) = \omega_{h_i}(s)$ for some $i \in [\lambda]$;
18: **end if**
19: **end for**
20: $S = \bigcup_{h \in H} S_h$, and let ω_S be a weight function defined on S, such that for each point $s \in S, \omega_S(s) = \omega_h(s)$ for some $h \in H$;
21: **return** (S, ω_S).

In Algorithm 1, the second random-solving stage is to solve the (k, z)-center problem in each B_h. Before starting the random-solving stage, we must address the challenging scenario where $|B_h| > s$. In case $|B_h| > s$, we distribute the points of $B_h \backslash \{h\}$ across consecutive machines to form sub-blocks $B_{h_1}, \ldots, B_{h_\lambda}$ and put the hub h in each machine, where the total number of points assigned to each machine remains less than s. This distribution can be efficiently managed using sorting and prefix sum within $O(1)$ rounds (Goodrich, Sitchinava, and Zhang 2011). Then, for each block B_h or sub-block B_{h_i}, we present the RS algorithm to solve the (k, z)-center problem, and record the weight of each opened center, as described in Algorithm 2. The basic idea behind the RS algorithm is to iteratively use a random ball covering technique to sample points as centers

Algorithm 2: RS $(R, \omega, h, r^*, m, \epsilon, \eta, \tau)$

Input: A set $R \subset \mathbb{R}^d$ of at most n points, a hub h, radius parameter r^*, machine parameter m, parameters $\epsilon, \eta \in (0,1)$, weight parameter τ, weight function ω.
Output: A set $G \subseteq R$ with weighted function $\omega_G : G \to \mathbb{R}^+$.

1: For each point $r \in R$, let $\omega(r)$ be its weight.
2: $G \leftarrow \{h\}, U \leftarrow R, Q = \emptyset, \kappa \leftarrow 0$;
3: **if** $|\{x \in U : d(h, x) \leq 4c_\rho r^*\}| \geq \tau$ **then**
4: Open it as center and set $U = U \backslash \{x \in U : d(h, x) \leq 4c_\rho r^*\}$;
5: **else**
6: Set $G \leftarrow \emptyset$;
7: **end if**
8: **while** $U \neq \emptyset$ and $\kappa < t = k \cdot \lceil \log_{1+\epsilon} \frac{mk}{\eta} \rceil$ **do**
9: $\kappa \leftarrow \kappa + 1$;
10: Sample c_κ from U uniformly and randomly;
11: **if** $\omega(c_\kappa) \geq \tau$ **then**
12: $C_\kappa \leftarrow \{x \in U : d(x, c_\kappa) \leq 4c_\rho r^*\}$;
13: $\vartheta(c_\kappa) \leftarrow \sum_{x \in C_\kappa} \omega(x)$;
14: $G \leftarrow G \cup \{c_\kappa\}$;
15: $U \leftarrow U \backslash C_\kappa$;
16: **end if**
17: **end while**
18: For each point $u \in U$, if $\omega(u) \geq \tau$, $G \leftarrow G \cup \{u\}$. Set $\vartheta(u) = \omega(u)$;
19: Define $\omega_G : G \to \mathbb{R}^+$ as a weight function of G and for each $g \in G$, $\omega_G(g) = \vartheta(g)$;
20: **return** (G, ω_G).

uniformly and randomly, ensuring most inliers are covered with a certain probability and the discarded outliers do not exceed ϵz. Initialize $G = \{h\}, U = R, Q = \emptyset$, and $\kappa = 0$. If the ball centered at h with radius $O(4c_\rho r^*)$ has a size larger than τ, open h as centers. Otherwise, set $G = \emptyset$. Algorithm 2 iteratively applies a random ball covering process to cover the optimal clusters in t rounds. In each round, the algorithm randomly and uniformly selects a point c_κ covers all points within a ball of radius $4c_\rho r^*$ centered at c_κ. If the size of this ball is larger than τ, c_κ is opened and assigned a new weight based on the number of points it covers. After t rounds of iteration, the remaining points in U are unclustered. If a point in U has a weight exceeding τ, add it to G. We show that by repeating the random ball covering process for $t = k \cdot \lceil \log_{1+\epsilon} \frac{mk}{\eta} \rceil$ times, with probability at least $1 - \eta$, the number of uncovered inliers in Algorithm 2 can be upper bound by ϵz.

Lemma 6. *By repeating at most $t = k \cdot \lceil \log_{1+\epsilon} \frac{mk}{\eta} \rceil$ rounds of steps 9-16 for Algorithm 2, with probability at least $1 - \eta$, Algorithm 2 obtains a $4c_\rho$-approximate solution S with at most ϵz inliers discarded when $\epsilon, \eta \in (0,1)$.*

Consider the case where $|Q| \geq s$ (the case when $|Q| < s$ is trivial). The ORS algorithm returns "FAIL" only if the set of hubs H is empty. Since each point in Q is added to H independently with probability p, the probability that the ORS algorithm returns "FAIL" is at most $(1-p)^{|Q|} \leq e^{-\Omega(p \cdot n^\delta)}$.

Then, we analyze the success probability of the ORS algorithm in the following lemma.

Lemma 7. *With probability parameter $p = \Omega\left(\frac{\log n}{n^\delta}\right)$, the success probability of Algorithm 1 is at least $1 - n^{-\Omega(1)}$.*

The following lemma gives the approximation guarantee, number of rounds and space complexity of Algorithm 1.

Lemma 8. *If Algorithm 1 does not report "FAIL", with probability at least $1 - \min\{e^{-\Omega(p \cdot n^\delta)}, n^{-\Omega(1)}\}$, Algorithm 1 can generate a set $S \subseteq Q$ of centers with $\nu(P, S) \leq 4c_\rho r^* = O(r^*)$ by discarding at most $(1+\epsilon)z$ outliers. Moreover, the algorithm can be executed in $O(1)$ rounds with local space $s = \Theta(n^\delta)$ and total space $\widetilde{O}(n^{1+\delta} \cdot \log \Delta)$, where c_ρ is a constant only depending on ρ.*

The following lemma provides an additional guarantee for Algorithm 1, showing that if at least one point of D_j^* is sampled as a hub and opened, the intersection size between the solution S from Algorithm 1 and any optimal cluster D_j^* can be bounded by $|H \cap D_j^*|$ with high probability. Let $P_{opt}^{r^*}$ denote an optimal partition of P that has cost at most r^*.

Lemma 9. *For any $D_j^* \subseteq P_{opt}^{r^*}$ ($j \in [k]$), with high probability, if at least one hub is selected from D_j^* and opened, then no further point $q \in D_j^* \backslash H$ is selected as a center. That is, $|S \cap D_j^*| = |H \cap D_j^*|$.*

Center-Reduction Algorithm

In the worst case, Algorithm 1 might be forced to open up to $\Theta(n^\delta \cdot m)$ centers across all machines. To reduce the number of opened centers, we propose a center-reduction algorithm as described in Algorithm 3. Algorithm 3 iteratively performs the ORS algorithm in $\Theta(\log \log n)$ rounds, reducing the intersections size between each optimal cluster $D_j^* (j \in [k])$ and the solution S to $\widetilde{O}(\log n)$. Each round of refinement introduces an additive error of $O(r^*)$ to the solution cost. For each point $p \in P$, let $\omega(p)$ be its weight. Initialize with $S_0 = P$, $p_0 = \Theta(\frac{\log n}{n^\delta})$, $t = n$, and $s_0 = t$. Suppose that we iteratively run the ORS algorithm in $\mathcal{L}$ rounds, where $\mathcal{L} = \Theta(\log \log n)$, where $s_i = \sqrt{s_{i-1}}$ and $p_i = \frac{1}{s_i}$. At the i-th ($i \in [\mathcal{L}]$) iteration, we apply the ORS algorithm to (S_{i-1}, ω_{i-1}) to obtain (S_i, ω_i), which serves as the input for the next iteration. A key observation is that the number of discarded inliers increases with each iteration. To address this, we introduce an (ϵ, η)-adaptive method in Algorithm 3 that controls the number of discarded inliers across m machines. This approach adjusts the sample size based on the data size, ensuring that for any iteration rounds $\mathcal{L}$, Algorithm 3 discards no more than ϵz inliers with constant probability. The lemma shows that by setting $\eta = \frac{\eta}{\mathcal{L}}$ and $\epsilon = \frac{\epsilon}{\mathcal{L}}$ in each iteration of the ORS algorithm, after $\mathcal{L}$ rounds, the number of discarded inliers is bounded by ϵz with probability at least $1 - \eta$.

Lemma 10. *Let $Z_{\mathcal{L}}$ denote the number of discarded inliers after these $\mathcal{L}$ iterations. By setting $\eta = \frac{\eta}{\mathcal{L}}$ and $\epsilon = \frac{\epsilon}{\mathcal{L}}$ in each iteration round of the ORS algorithm, after $\mathcal{L}$ iterations, $Z_{\mathcal{L}} \leq \epsilon z$ holds with probability at least $1 - \eta$, where $\eta, \epsilon \in (0, 1)$.*

Algorithm 3: CR $(P, \omega, r^*, m, s, \epsilon, \eta, t, z, \rho, \tau)$

Input: A set $P \subset \mathbb{R}^d$ of n points among m machines with local space s, outliers parameter z, radius parameter $r^* \in \mathbb{R}^+$, frequency parameters $t \leq n$, LSH parameter $\rho \in (0, 1)$, parameters $\epsilon, \eta \in (0, 1)$, weight parameter τ, weight function ω.

Output: A set $S \subseteq P$ with a weighted function $\omega_S : S \to \mathbb{R}^+$.

1: For each point $p \in P$, let $\omega(p)$ be its weight;
2: $p_0 = \Theta(\frac{\log n}{n^\delta})$, $t = n$, $s_0 = t$, $S_0 \leftarrow P$;
3: **for** $i = 1$ to $\mathcal{L} = \Theta(\log \log t)$ **do**
4: $\quad (S_i, \omega_i) \leftarrow$ ORS $(S_{i-1}, \omega_{i-1}, p_{i-1}, r^*, m, s, \frac{\epsilon}{\mathcal{L}}, \frac{\eta}{\mathcal{L}}, z, \rho, \tau)$;
5: $\quad s_i \leftarrow \sqrt{s_{i-1}}$ and $p_i = \frac{1}{s_i}$;
6: **end for**
7: **return** $(S = S_{\mathcal{L}}, \omega_S = \omega_{\mathcal{L}})$.

The following lemma provides the approximation guarantee, number of rounds and space complexity of Algorithm 3, which can be obtained from Lemma 8.

Lemma 11. *Algorithm 3 can be completed in $O(\log \log n)$ MPC rounds, using local space $s = \Theta(n^\delta)$ and total space $\widetilde{O}(n^{1+\delta} \cdot \log \Delta)$. It outputs a solution S with $\nu(Q_t, S) = O(\log \log n \cdot r^*)$.*

We show the key property of Algorithm 3 using induction. By setting $\mathcal{L} = \Theta(\log \log n)$, the intersection size between each optimal cluster D_j^* and solution $S_{\mathcal{L}}$ is reduced to $O(\log n \cdot (\log \log n)^2)$ with high probability, ensuring an additive error of $O(r^*)$ per round in the cost of the solution.

Lemma 12. *For a partition $P_{opt}^{r^*}$ of P that has cost r^*, Algorithm 3 outputs a solution $S_{\mathcal{L}} \subseteq P$ such that for any optimal cluster $D_j^* \subseteq P_{opt}^{r^*}$ ($j \in [k]$), if $|D_j^* \cap P| \leq n$, with probability at least $1 - \frac{1}{n^{\Omega(1)}}$, $|D_j^* \cap S_{\mathcal{L}}| = O(\log n \cdot (\log \log n)^2)$.*

During the $\Theta(\log \log n)$ rounds, some optimal clusters referred to as badly optimal clusters may shrink below $\widetilde{O}(\log n)$ in size before the rounds are completed, making their points cannot be sampled as hubs with high probability. This requires Algorithm 2 to cover them, potentially discarding some inliers. To address challenges in distinguishing inliers from outliers when they are closely intermixed, we assume that each outlier is at least $O(4c_\rho r^*)$ away from any inlier. Under this assumption, we introduce a τ-weight truncation method, where in each iteration of Algorithm 3, point weights are computed as described at step 13 of Algorithm 2, discarding points with weights smaller than τ, as detailed in step 18. The following lemmas show how τ affects discarded outliers and opened centers per iteration.

Lemma 13. *In each iteration of Algorithm 3, the number of outliers that can be selected as centers is bounded by $\frac{z}{\tau}$.*

Lemma 14. *Assume that the distance between each outlier and inlier is larger than $O(4c_\rho r^*)$. After Algorithm 3, the number of outliers that can be selected as centers is bounded by $\frac{z}{\tau} \cdot \mathcal{L}$, and the number of discarded inliers is bounded by $\widetilde{O}(\log n) k\tau$.*

Then, we analyze how the value of τ affects the number of discarded inliers and opened centers in $\Theta(\log\log n)$ rounds with additional assumptions.

Lemma 15. *After Algorithm 3, by setting $\tau = \frac{\epsilon z}{k\widetilde{O}(\log n)}$, with the assumption that the distance between each outlier and each inlier is larger than $\widetilde{O}(4c_\rho r^*)$ and $z \geq \frac{k\widetilde{O}(\log n)}{\epsilon}$, the number of outliers that can be selected as centers is bounded by $\frac{k\widetilde{O}(\log n)}{\epsilon}$, and the number of discarded inliers is bounded by ϵz.*

MPC-k-Outliers Algorithm

In this section, we propose the MPCKO algorithm as described in Algorithm 4. The basic idea is to use a two-stage reduction strategy to iteratively reduce the number of opened centers in the optimal clusters. In the first stage (steps 3-7 of Algorithm 4), an iteration of the CR algorithm is used to reduce the size of the intersection between nearly all optimal clusters and the solution S_i. At the i-th iteration, the size of the intersection between nearly all optimal clusters and the solution S_i can be reduced to $\widetilde{O}(\log^{(i)} n)$ after $\Theta(\log^{(i+1)} n)$ rounds, with each round adding an error of $\frac{r^*}{\log^{(i+1)} n}$ to the cost of the solution. This process continues until the sizes of nearly all optimal clusters are reduced to $\widetilde{O}(\log^* n)$. However, during the iteration process, there may be some optimal clusters whose sizes cannot be reduced to $\widetilde{O}(\log^* n)$, which is a much more complicated case. Hence, in the second stage (steps 9-10 of Algorithm 4), an iteration of the ORS algorithm is used to address this complex scenario, where the size of these challenging optimal clusters decreases by a constant factor with each successive round, introducing an additive error of $O(r^*)$ at each round.

In **Stage 1**, a crucial parameter α is defined to account for the number of iterations. Algorithm 4 applies the CR algorithm $\alpha = \Theta(\log^* n)$ times. For each point $p \in P$, let $\omega(p) = 1$ be its weight. Initialize with $S_0 = P$, $\mathcal{L}_0 = \log\log n$, $r_0 = \frac{r^*}{\log\log n}$, and $t_0 = n$. In the i-th iteration of **Stage 1**, Algorithm 4 applies the CR algorithm using (S_{i-1}, ω_{i-1}) as input and producing (S_i, ω_i) as the output, where $t_i = \widetilde{\Theta}(\log^{(i)} n)$ and $r_i = \frac{r^*}{\log\log t_i}$. In **Stage 2**, Algorithm 4 applies the ORS algorithm β times, where $\beta = \Theta(\log^{(\alpha+1)} n)$ and α is the iteration parameter of **Stage 1**. The input of **Stage 2** is $(S_\alpha, \omega_\alpha)$. In the j-th iteration of **Stage 2**, Algorithm 4 runs ORS algorithm using $(S_{\alpha+j-1}, \omega_{\alpha+j-1})$ as input and producing $(S_{\alpha+j}, \omega_{\alpha+j})$ as the output. Denote $(S_{\alpha+\beta}, \omega_{\alpha+\beta})$ as the output of Algorithm 4. The following lemma provides the approximation guarantee, number of rounds and space complexity of Algorithm 4. Let r^* be the optimal radius, determined using $\log\Delta$ estimates, ensuring a constant factor approximation loss. Thus, the total space of our MPC algorithm is $\widetilde{O}(n^{1+\delta}\cdot\log^2\Delta)$.

Lemma 16. *Algorithm 4 runs in $O(\log\log n)$ MPC rounds with local space $s = \Theta(n^\delta)$ and total space $\widetilde{O}(n^{1+\delta}\cdot\log^2\Delta)$. When $\alpha = \Theta(\log^* n)$, it produces a solution S with $\nu(Q_t, S) = O(r^*\cdot\log^* n)$.*

Algorithm 4: MPCKO$(P, r^*, m, s, \epsilon, \eta, z, \rho, \tau, \omega)$

Input: A set $P \subset \mathbb{R}^d$ of n points among m machines with local space s, outliers parameter $z \in [0, n]$, radius parameter $r^* \in \mathbb{R}^+$, LSH parameters $\rho \in (0,1)$, parameters $\epsilon, \eta \in (0,1)$, weight parameter τ, weight function ω.
Output: A set $S \subseteq P$ with a weighted function $\omega_S : S \to \mathbb{R}^+$.

1: For each point $p \in P$, let $\omega(p) = 1$ be its weight;
2: $S_0 \leftarrow P$, $t_0 = n$, $\mathcal{L}_0 = \log\log n$, and $r_0 = \frac{r^*}{\log\log n}$;
Stage 1:
3: **for** $i = 1$ to $\alpha = \Theta(\log^* n)$ **do**
4: $\quad\mathcal{L}_i = \log\log t_{i-1}$;
5: $\quad(S_i, \omega_i) \leftarrow$ CR $(S_{i-1}, \omega_{i-1}, r_{i-1}, t_{i-1}, m, s, \frac{\epsilon}{\alpha\mathcal{L}_i}, \frac{\eta}{\alpha\mathcal{L}_i}, z, \rho, \tau)$;
6: $\quad t_i = \Theta(\log t_{i-1}\cdot(\log\log t_{i-1})^{d+2}) = \widetilde{\Theta}\left(\log^{(i)} n\right)$;
7: $\quad r_i = \frac{r^*}{\log\log t_i}$;
8: **end for**
Stage 2:
9: **for** $j = 1$ to $\beta = \Theta(\log^{(\alpha+1)} n)$ **do**
10: $\quad(S_{\alpha+j}, \omega_{\alpha+j}) \leftarrow$ORS$(S_{\alpha+j-1}, \omega_{\alpha+j-1}, m, s, \frac{1}{2}, r^*, \frac{\epsilon}{\beta}, \frac{\eta}{\beta}, z, \rho, \tau)$;
11: **end for**
12: **return** $(S = S_{\alpha+\beta}, \omega_S = \omega_{\alpha+\beta})$.

In Lemma 17, by setting $\alpha = \Theta(\log^* n)$, the number of centers opened in the intersection between all optimal clusters and the solution of Algorithm 4 is bounded by $k(1+o(1))$.

Lemma 17. *With probability at least $1 - \frac{1}{(\log^{(\alpha-1)} n)^{\Omega(1)}}$, Algorithm 4 reduces the number of opened centers in the intersection between all optimal clusters and $S_{\alpha+\beta}$ to $|S^{opt}_{\alpha+\beta}| \leq |\mathcal{C}_{r^*}|\left(1 + \frac{1}{\widetilde{\Theta}(\log^{(\alpha)} n)}\right) + \widetilde{\Theta}((\log^{(\alpha)} n)^3)$, where $\mathcal{C}_{r^*}$ is a clustering of P that has the minimum number of centers among all possible clusterings of P such that the cost is at most r^*. Moreover, $|\mathcal{C}_{r^*}| = \Omega((\log n)^c)$, and $c \geq 1$ is a constant. By setting $\alpha = \Theta(\log^* n)$, $|S^{opt}_{\alpha+\beta}| \leq k(1+o(1))$.*

Using Lemma 15, we will analyze the number of opened centers including outliers in Algorithm 4.

Lemma 18. *Assuming each outlier is at least $\widetilde{O}(4c_\rho r^*)$ away from any inlier and $z \geq \frac{k\widetilde{O}(\log n)}{\epsilon}$, the number of opened centers in Algorithm 4 is bounded by $(k(1+o(1)) + \frac{k\widetilde{O}(\log n)}{\epsilon}$. Additionally, the number of discarded outliers is bounded by $(1+\epsilon)z$.*

Conclusion

In this paper, we introduce the first fully-scalable $O(\log^* n)$-approximation algorithm for the MPC (k, z)-center problem, which runs in $O(\log\log n)$ rounds and operates with local space $s = \Theta(n^\delta)$ and total space $\widetilde{O}(n^{1+\delta}\cdot\log^2\Delta)$. Our proposed approach covers most inliers and ensures that the number of discarded outliers is bounded by $(1+\epsilon)z$.

Acknowledgments

This work was supported by National Natural Science Foundation of China (62432016, 62172446), Open Project of Xiangjiang Laboratory (22XJ02002), and Central South University Research Program of Advanced Interdisciplinary Studies (2023QYJC023).

References

Amagata, D. 2024. Fair k-center clustering with outliers. In *Proc. 27th International Conference on Artificial Intelligence and Statistics*, 10–18.

Bachem, O.; Lucic, M.; and Krause, A. 2018. Scalable k-means clustering via lightweight coresets. In *Proc. 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1119–1127.

Bateni, M.; Esfandiari, H.; Fischer, M.; and Mirrokni, V. S. 2021. Extreme k-center clustering. In *Proc. 35th Conference on Artificial Intelligence*, 3941–3949.

Braverman, V.; Cohen-Addad, V.; Jiang, S. H.; Krauthgamer, R.; Schwiegelshohn, C.; Toftrup, M. B.; and Wu, X. 2022. The power of uniform sampling for coresets. In *Proc. 63rd IEEE Annual Symposium on Foundations of Computer Science*, 462–473.

Cao, N.; Huang, S.-E.; and Su, H.-H. 2024. Breaking 3-factor approximation for correlation clustering in polylogarithmic rounds. In *Proc. 35th Annual ACM-SIAM Symposium on Discrete Algorithms*, 4124–4154.

Ceccarello, M.; Pietracaprina, A.; and Pucci, G. 2019. Solving k-center clustering (with outliers) in MapReduce and streaming, almost as Accurately as Sequentially. *Proc. VLDB Endow.*, 12(7): 766–778.

Chakrabarty, D.; Goyal, P.; and Krishnaswamy, R. 2016. The non-uniform k-center problem. In *Proc. 43rd International Colloquium on Automata, Languages, and Programming*, 1–15.

Charikar, M.; Khuller, S.; Mount, D. M.; and Narasimhan, G. 2001. Algorithms for facility location problems with outliers. In *Proc. 12th Annual Symposium on Discrete Algorithms*, 642–651.

Chen, J.; Azer, E. S.; and Zhang, Q. 2018. A practical algorithm for distributed clustering and outlier detection. In *Proc. 32rd Advances in Neural Information Processing Systems*, 2253–2262.

Cohen-Addad, V.; Larsen, K. G.; Saulpic, D.; and Schwiegelshohn, C. 2022. Towards optimal lower bounds for k-median and k-means coresets. In *Proc. 54th Annual ACM SIGACT Symposium on Theory of Computing*, 1038–1051.

Cohen-Addad, V.; Mirrokni, V.; and Zhong, P. 2022. Massively parallel k-means clustering for perturbation resilient instances. In *Proc. 39th International Conference on Machine Learning*, 4180–4201.

Cohen-Addad, V.; Saulpic, D.; and Schwiegelshohn, C. 2021. A new coreset framework for clustering. In *Proc. 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 169–182.

Coy, S.; Czumaj, A.; and Mishra, G. 2023. On parallel k-center clustering. In *Proc. 35th ACM Symposium on Parallelism in Algorithms and Architectures*, 65–75.

de Berg, M.; Biabani, L.; and Monemizadeh, M. 2023. k-center clustering with outliers in the MPC and streaming model. In *Proc. 37th IEEE International Parallel and Distributed Processing Symposium*, 853–863.

Dean, J.; and Ghemawat, S. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1): 107–113.

Delic, A.; Grcic, M.; and Segvic, S. 2024. Outlier detection by ensembling uncertainty with negative objectness. *CoRR*, abs/2402.15374.

Ding, H.; Liu, Y.; Huang, L.; and Li, J. 2016. k-means clustering with distributed dimensions. In *Proc. 33rd International Conference on Machine Learning*, 1339–1348.

Ding, H.; Yu, H.; and Wang, Z. 2019. Greedy strategy works for k-center clustering with outliers and coreset construction. In *Proc. 27th Annual European Symposium on Algorithms*, 40:1–40:16.

Feder, T.; and Greene, D. H. 1988. Optimal algorithms for approximate clustering. In *Proc. 20th Annual ACM Symposium on Theory of Computing*, 434–444.

Gonzalez, T. F. 1985. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38: 293–306.

Goodrich, M. T.; Sitchinava, N.; and Zhang, Q. 2011. Sorting, searching, and simulation in the MapReduce framework. In *Proc. 22nd International Symposium Algorithms and Computation*, 374–383.

Guha, S.; Li, Y.; and Zhang, Q. 2017. Distributed partial clustering. In *Proc. 29th ACM Symposium on Parallelism in Algorithms and Architectures*, 143–152.

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

Im, S.; Kumar, R.; Lattanzi, S.; Moseley, B.; and Vassilvitskii, S. 2023. Massively parallel computation: algorithms and applications. *Found. Trends Optim.*, 5(4): 340–417.

Isard, M.; Budiu, M.; Yu, Y.; Birrell, A.; and Fetterly, D. 2007. Dryad: distributed data-parallel programs from sequential building blocks. In *Proc. 2nd European Conference on Computer Systems*, 59–72.

Jiang, X.; Ma, J.; Jiang, J.; and Guo, X. 2019. Robust feature matching using spatial clustering with heavy outliers. *IEEE Transactions on Image Processing*, 29: 736–746.

Kanjanawattana, S. 2019. A novel outlier detection applied to an adaptive k-means. *International Journal of Machine Learning and Computing*, 9(5): 569–574.

Karimireddy, S. P.; Kale, S.; Mohri, M.; Reddi, S. J.; Stich, S. U.; and Suresh, A. T. 2020. SCAFFOLD: stochastic controlled averaging for federated learning. In *Proc. 37th International Conference on Machine Learning*, 5132–5143.

Li, S.; and Guo, X. 2018. Distributed k-clustering for data with heavy noise. In *Proc. 32nd Advances in Neural Information Processing Systems*, 7849–7857.

Malkomes, G.; Kusner, M. J.; Chen, W.; Weinberger, K. Q.; and Moseley, B. 2015. Fast distributed k-center clustering with outliers on massive data. In *Proc. 29th Advances in Neural Information Processing Systems*, 1063–1071.

Megiddo, N.; and Supowit, K. J. 1984. On the complexity of some common geometric location problems. *SIAM J. Comput.*, 13(1): 182–196.

White, T. 2015. *Hadoop-the definitive guide: storage and analysis at internet scale*. O'Reilly.

Zaharia, M.; Chowdhury, M.; Franklin, M. J.; Shenker, S.; and Stoica, I. 2010. Spark: cluster computing with working sets. In *Proc. 2nd USENIX Workshop on Hot Topics in Cloud Computing*.