Massively Parallel k-Means Clustering for Perturbation Resilient Instances

Vincent Cohen-Addad [*1] **Vahab Mirrokni** [*1] **Peilin Zhong** [*1]

Abstract

We consider k-means clustering of n data points in Euclidean space in the Massively Parallel Computation (MPC) model, a computational model which is an abstraction of modern massively parallel computing system such as MapReduce. Recent work provides evidence that getting $O(1)$-approximate k-means solution for general input points using $o(\log n)$ rounds in the MPC model may be impossible under certain conditions [Ghaffari, Kuhn & Uitto'2019]. However, the real-world data points usually have better structures. One instance of interest is the set of data points which is perturbation resilient [Bilu & Linial'2010]. In particular, a point set is α-perturbation resilient for k-means if perturbing pairwise distances by multiplicative factors in the range $[1, \alpha]$ does not change the optimum k-means clusters. We bypass the worst case lower bound by considering the perturbation resilient input points and showing $o(\log n)$ rounds k-means clustering algorithms for these instances in the MPC model. Specifically, we show a fully scalable $(1+\varepsilon)$-approximate k-means clustering algorithm for $O(\alpha)$-perturbation resilient instance in the MPC model using $O(1)$ rounds and $O_{\varepsilon,d}(n^{1+1/\alpha^2+o(1)})$ total space. If the space per machine is sufficiently larger than k, i.e., at least $k \cdot n^{\Omega(1)}$, we also develop an optimal k-means clustering algorithm for $O(\alpha)$-perturbation resilient instance in MPC using $O(1)$ rounds and $O_d(n^{1+o(1)} \cdot (n^{1/\alpha^2} + k))$ total space.

1. Introduction

Metric clustering is at the heart of data analysis and modern unsupervised machine learning with many applications for mining massive datasets. As a canonical clustering problem, k-means clustering is defined as follows: Given a dataset where each data element is represented by a vector of real-valued features, the goal is to find k representative vectors, called *centers*, such that the sum of squared distances from each input vector to the closest center is minimized. Similarly, in k-median clustering, the goal is to minimize sum of distances to closest centers. k-means and k-median have become essential building blocks for unveiling hidden patterns and extracting information in datasets, especially in the unsupervised clustering contexts where supervised machine learning cannot be applied, or little is known about the data or when the dataset is massive and the competitive supervised methods become impractical.

These clustering problems have received a lot of attention through the years and for almost half a century. Hence, good progress on our understanding of the complexity of the problems has been made. We know that the problems are hard to approximate within a factor better than 1.1 in high-dimensional Euclidean spaces and admits approximation schemes in low-dimension (Arora et al., 1998; Kolliopoulos & Rao, 2007; Cohen-Addad, 2018; Cohen-Addad et al., 2019a). We further know of constant-factor approximation algorithms for high-dimensional Euclidean spaces, better than for general metric spaces (Cohen-Addad et al., 2019b). Also due to their applications, these problem have been studied extensively from both theoretical and practical perspectives with many approximation algorithms and heuristics (Lloyd, 1982; Arthur & Vassilvitskii, 2007; Byrka et al., 2014; Kanungo et al., 2004; Jain et al., 2003; Li, 2011).

The era of massive datasets has naturally led researchers to design massively parallel approximation algorithms, and in particular in the Massively Parallel Computation (MPC) model. This model has become the main model for analyzing algorithms for large-scale parallel computing (Karloff et al., 2010; Goodrich et al., 2011; Beame et al., 2013) and serves as a theoretical abstraction of real-world systems like MapReduce (Dean & Ghemawat, 2008), Hadoop (White, 2012) and Spark (Zaharia et al., 2010). The goal is to design distributed algorithms that employ machines with sublinear space, and also run in a sublinear (and hopefully sublogarithmic) number of rounds of computation. In addition to round and memory complexity, this model takes into account the total work (including total communication) as an important

[*]Equal contribution [1]Google Research. Correspondence to: Vincent Cohen-Addad <cohenaddad@google.com>, Vahab Mirrokni <mirrokni@google.com>, Peilin Zhong <peilinz@google.com>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

factor for the quality of the algorithm.

k-Means and related clustering problems are very well studied in distributed, parallel, and streaming settings (Balcan et al., 2013; Bahmani et al., 2012; Bachem et al., 2015; Lucic et al., 2016; Bachem et al., 2016; 2018; Bhaskara & Wijewardena, 2018a). For example, as a well-established technique in designing distributed algorithms, core-sets have been extensively studied (Agarwal et al., 2004) for k-means and other clustering problems (Malkomes et al., 2015; Ceccarello et al., 2018; Bachem et al., 2015; Lucic et al., 2016; Bachem et al., 2016; 2018; Bhaskara & Wijewardena, 2018a). However, all these results suffer from two shortcomings: They either run in $\Omega(\log n)$ number of rounds of computation, or need $\Omega(k)$ space per machine. Moreover, recent results provide evidence that designing a fully scalable[1] MPC constant-factor approximation algorithm with $o(\log n)$ rounds for these problems may be impossible under certain complexity assumptions (Ghaffari et al., 2019). These shortcomings can be quite unsatisfactory in real-world applications: first of all, with billions of points, $\Omega(\log n)$ rounds of synchronization can result in prohibitive running time. Secondly, in applications like detecting duplicates or compression, k could be very large (e.g., of the order of 100s of millions), and we may not have enough space to compute a solution of size $\Omega(k)$ on a single machine. The importance of designing algorithms with $o(k)$ space per machine has been observed recently in several papers (Bhaskara & Wijewardena, 2018a; Epasto et al., 2019; Bateni et al., 2021). In particular, the only known massively parallel algorithm for k-means (or k-median) that does not require $\Omega(k)$ memory per machine is due to Bhaskara et al. (Bhaskara & Wijewardena, 2018b) who developed a *bicriteria* approximation algorithm with approximation factor $O((\log n \log\log n)^2)$ using $O(k \log k \log n)$ centers instead of k, a memory per machine of $s \in \Omega(d \log n)$, and $O(\log_s n)$ parallel rounds. While the trade-off between memory-per-machine and number of rounds may be satisfactory, the approximation quality and the fact that the solution outputs many more centers than k is undesirable in practice. Indeed, in several applications where k is large, outputting $k \log k \log n$ centers may lead to clusters of extremely small size and somewhat defeat the purpose of the clustering approach.

To bypass the above barriers and significantly improve over state-of-the-art approaches, we need to go *beyond the worst-case*. Designing and analysing algorithms in so called beyond worst-case instances has been successful in the past to shed lights on popular heuristics such as k-means++, Lloyd or local search. While the above methods can perform poorly in the worst-case, researchers have shown that when the underlying instance exhibits a "ground-truth" clustering structure, then the methods may actually be able to identify the optimum clustering, hence bypassing NP-hardness, and even APX-hardness barriers.

A class of non-worst-case instances that is heavily studied in the clustering literature is the class of *perturbation resilient* instances. Roughly speaking, the optimal clusters of an α-perturbation-resilient input instance are stable when pairwise distances of input points are perturbed by a multiplicative factor at most α. For Euclidean k-median and k-means, Awasthi et al. (Awasthi et al., 2012) obtained a polynomial-time optimal clustering algorithm when the instance is 3-perturbation-resilient. This was later improved to $(1+\sqrt{2})$-perturbation-resilient by Balcan and Liang (Balcan & Liang, 2016) and to 2-perturbation-resilient by Angelidakis et al. (Angelidakis et al., 2017). It is NP-hard to compute an optimum solution for $(2-\varepsilon)$-perturbation-resilient instances. Cohen-Addad and Schwiegelshohn (Cohen-Addad & Schwiegelshohn, 2017) also showed that local search method finds the optimal clusters in polynomial time for $(3+\varepsilon)$-perturbation-resilient instances. However, none of above algorithms can be implemented efficiently in the MPC model (or even in near-linear time in the sequential computing model). k-Median was studied by (Voevodski, 2021) in parallel computing model, but it requires $\Omega(k)$ space per machine and may not find the optimal solution (or even $(1+\varepsilon)$-approximation).

1.1. Our Results and Techniques

In this work, we show that when the instance indeed exhibits a "ground-truth" clustering structure, as captured by the popular notion of α-perturbation-resilience, then one can design efficient algorithms that solve the k-median or k-means problems exactly in $O(1)$ rounds, and with memory per machine $O(n^\delta k)$ and total memory $(nd \log n) + n^{1+O(1/\alpha^2)+o(1)} + nk$. More importantly, there exists $(1+o(1))$-approximation algorithms for the problem that runs in $O(1)$ rounds and with arbitrary memory per machine, as long as the total memory is $(nd \log n) + n^{1+O(1/\alpha^2)+o(1)}$.

At a high level, our algorithms first compute a hierarchical clustering tree and then run a dynamic programming over the tree to find optimal clusters. Although various hierarchical clustering trees for perturbation-resilient instances were proposed for better sequential algorithms, their constructions are all based on the linkage operations (Balcan & Liang, 2016; Awasthi et al., 2012) which are in the flavor of greedy algorithms and thus make the construction hard to be parallelized. In contrast, we exploit the structural properties of the optimal clusters of perturbation-resilient instances and prove that the optimal clusters can be well-captured by locality sensitive hashing (LSH). By applying LSH of various scales simultaneously, we are able to construct all

[1] An MPC algorithm is fully scalable if it works for any local memory size, in particular, it should work when local memory less than k for $k = n^{O(1)}$.

layers of the clustering tree at the same time and thus the tree construction step only takes $O(1)$ rounds. Another challenging part is to parallelize dynamic programming. The best previously known parallel dynamic programming algorithm over general trees needs $\Omega(\log n)$ rounds (Bateni et al., 2018). However, surprisingly, we show that if the depth of the tree is small and the tree edges are stored on machines in a well-organized manner, we are able to solve a class of dynamic programming problems over trees in $O(1)$ rounds (see Algorithm 7). To achieve this, we develop a novel task scheduling process via subtree generation. Finally, to remove the dependence of space on k, instead of running dynamic programming over cluster sizes to optimize the clustering cost, we run dynamic programming over discretized clustering cost and try to minimize the number of centers needed to achieve that cost. This only increases the clustering cost by a $(1+o(1))$-factor.

Theorem 1.1 (Exact k-means)**.** *Let $\alpha > 6.1$. Let $\delta \in (0,1)$ be an arbitrary constant. Consider a set $P \subset \mathbb{R}^d$ of n points which is α-perturbation resilient to k-means and the aspect ratio of P is at most $2^{n^{o(1)}}$[2]. There is an MPC algorithm which outputs the optimal k-means clustering of P with probability at least $1-1/\mathrm{poly}(n)$ using $O(1)$ rounds and $O(nd\log n)+n^{1+o(1)}(k+n^{O(1/\alpha^2)})$ total space. The memory needed per machine is $\Theta(n^\delta k)$.*

Theorem 1.2 (Approximate k-means)**.** *Let $\alpha > 6.1$ and $\varepsilon \in (1/n^{o(1)}, 0.5)$. Consider a set $P \subset \mathbb{R}^d$ of n points which is α-perturbation resilient to k-means and the aspect ratio of P is at most $2^{n^{o(1)}}$. There is a fully scalable MPC algorithm which outputs the $(1+\varepsilon)$-approximate k-means clustering of P with probability at least $1-1/\mathrm{poly}(n)$ using $O(1)$ rounds and $O(nd\log n)+n^{1+O(1/\alpha^2)+o(1)}$ total space.*

We note that the above can be extended to the k-median problem. The challenge in extending this to the k-median problem is that the geometric median, a.k.a. the Fermat-Weber point, is not known to be computable even in the offline setting in polynomial time. Thus, as usual with the k-median problem one must resort to $(1+\varepsilon)$ approximation. To do this in parallel it is enough for each cluster to sample $\mathrm{poly}(d/\epsilon)$ points of the cluster and run a $(1+\varepsilon)$-approximation algorithm for computing the geometric median of the samples. This can be done in $O(1)$ rounds and leads to a median that generalizes to the entire cluster up to a factor $(1+\varepsilon)$, see Appendix D for more discussions.

Finally, we present an empirical study of algorithms validating their effectiveness.

[2]See Section 2 for the definition of aspect ratio. It is common to assume that the aspect ratio is $\mathrm{poly}(n)$ since each coordinate of a point is represented by $O(\log n)$ bits (one word) in practice. Our result is more general since it works even when the aspect ratio is an arbitrary $2^{n^{o(1)}}$ which can be much greater than $\mathrm{poly}(n)$.

2. Preliminaries and Notation

We define $[n] := \{1, 2, \cdots, n\}$. For any two points $p, q \in \mathbb{R}^d$, we use $\mathrm{dist}(p,q)$ to denote their Euclidean distance, i.e., $\mathrm{dist}(p,q) := \|p-q\|_2$. The aspect ratio of P is $\frac{\max_{p,q\in P}\mathrm{dist}(p,q)}{\min_{p\neq q\in P}\mathrm{dist}(p,q)}$. We consider k-means clustering over a set $P \subset \mathbb{R}^d$ of n points. The goal is to partition P into k disjoint clusters $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$ and assign each cluster C_i a center z_i such that $\sum_{i=1}^k \sum_{p\in C_i} \mathrm{dist}^2(p, z_i)$ is minimized. A well-known fact is that a center z is optimal for a cluster C when z is the mean of the cluster points, i.e., $z = \mathrm{mean}(C) := (\sum_{p\in C} p)/|C|$. Thus, we can define the cost of clustering $\mathcal{C}$ as $\mathrm{cost}(\mathcal{C}) := \sum_{i=1}^k \sum_{p\in C_i} \mathrm{dist}^2(p, \mathrm{mean}(C_i))$, and the goal of k-means is to find clustering $\mathcal{C}$ minimizing $\mathrm{cost}(\mathcal{C})$. Suppose $\mathcal{C}^*$ is the optimal k-means clustering of P. If $\mathcal{C}$ is a partition of P with $|\mathcal{C}| \le k$ and $\mathrm{cost}(\mathcal{C}) \le \beta \cdot \mathrm{cost}(\mathcal{C}^*)$, then we say that $\mathcal{C}$ is a β-approximate k-means clustering of P.

Definition 2.1. A point set $P = \{p_1, p_2, \cdots, p_n\}$ is α-perturbation resilient to k-means if for any point set $P' = \{p'_1, p'_2, \cdots, p'_n\}$ satisfying $\forall i, j \in [n], \mathrm{dist}(p_i, p_j) \le \mathrm{dist}(p'_i, p'_j) \le \alpha\,\mathrm{dist}(p_i, p_j)$, there is a unique optimal k-means clustering $\mathcal{C}' = \{C'_1, C'_2, \cdots, C'_k\}$ such that $\forall i \in [n], j \in [k], p'_i \in C'_j \iff p_i \in C_j$, where $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$ is the optimal k-means clustering for P.

The massively parallel computation (MPC) model. The MPC model (Karloff et al., 2010; Beame et al., 2017; Goodrich et al., 2011) is an abstraction of modern massively parallel computing systems such as MapReduce (Dean & Ghemawat, 2008). In the MPC model, there are p machines each with local memory size s which is sublinear in the input size. Before computation starts, the input is distributed arbitrarily on some input machines. The computation proceeds in rounds. In each round, each machine performs local computations based on the data in its local memory. At the end of a round, each machine can send messages to arbitrary other machines. But the total size of messages sent/received by a machine in one round is at most $O(s)$. At the end of computation, the output is distributed on some output machines. The main goal is to obtain an MPC algorithm with small *parallel time* (number of rounds) We also want the total space $p \cdot s$ needed by the algorithm to be as small as possible. Furthermore, scalability is also considered in many applications. In particular, we want the algorithm to work even when the local memory s is small. Suppose the input size is N. If the algorithm works for $s = O(N^\delta)$ for any constant $\delta \in (0,1)$, then the algorithm is fully scalable. There are many fully scalable MPC primitives. The most basic one is sorting.

Theorem 2.2 ((Goodrich et al., 2011; Goodrich, 1999))**.** *There is a fully scalable MPC algorithm which sorts N data items in $O(1)$ rounds using $O(N)$ total space.*

Parallel random access machine (PRAM) (Gibbons & Rytter, 1989) is a classic model of parallel computing. Based on MPC sorting algorithm, (Goodrich et al., 2011) shows that any PRAM algorithm can be simulated by a fully scalable MPC algorithm with the same number of rounds. Thus, basic PRAM operations can also be implemented by fully scalable MPC algorithms. We refer readers to Section E of (Andoni et al., 2018) for more basic MPC primitives.

3. Offline Algorithms for α-Perturbation Resilient Point Set

In this section, we describe (approximate) k-means clustering algorithms for α-perturbation resilient point set in offline setting. In later sections, we will show their implementations in the massively parallel computation setting. We put missing proofs and other details in Appendix B.

3.1. Structural Properties

Let us show some structural properties of α-perturbation resilient point set.

Definition 3.1. A point set P satisfies α-center proximity property for k-means if for any clusters $C_i \neq C_j \in \mathcal{C}$ where $\mathcal{C}$ is the optimal k-means clustering for P, any point $p \in C_i$ satisfies $\alpha \operatorname{dist}(p, \operatorname{mean}(C_i)) < \operatorname{dist}(p, \operatorname{mean}(C_j))$.

Lemma 3.2 (Lemma 3.2 of (Balcan & Liang, 2016))**.** *Any point set P which is α-perturbation resilient to k-means also satisfies α-center proximity for k-means.*

Lemma 3.3. *Consider a point set P which satisfies α-center proximity property for k-means. Let $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$ be the optimal k-means clustering for P. Consider an arbitrary cluster C_i and let $D := \max_{x,y \in C_i} \operatorname{dist}(x, y)$. Then, $(\alpha/4 - 1/2)D < \min_{p \in C_i, s \in P \setminus C_i} \operatorname{dist}(p, s)$.*

3.2. Near Neighbor Graph via LSH

An important tool is locality sensitive hashing (LSH).

Definition 3.4 (LSH)**.** Consider a family $\mathcal{H}$ of hash functions mapping $\mathbb{R}^d$ into some universe U. $\mathcal{H}$ is $(D, cD, \mathbf{p}_1, \mathbf{p}_2)$-sensitive, if $\forall x, y \in \mathbb{R}^d$, it satisfies: (1). If $\operatorname{dist}(x, y) \leq D$, $\Pr_{h \in \mathcal{H}}[h(x) = h(y)] \geq \mathbf{p}_1$. (2) If $\operatorname{dist}(x, y) \geq cD, \Pr_{h \in \mathcal{H}}[h(x) = h(y)] \leq \mathbf{p}_2$.

There are various constructions of LSH (see e.g., (Datar et al., 2004; Andoni & Indyk, 2006)). In particular, (Andoni & Indyk, 2006) shows how to construct $(D, cD, 1/n^{O(1/c^2)+o(1)}, 1/n^{\Omega(1)})$-sensitive family of hashing functions for any $D > 0, c > 1$ and dimension $d > 0$. Similar to (Balcan & Liang, 2016), we can use LSH to construct near neighbor graph (see Algorithm 1).

Lemma 3.5. *Consider a set $P \subset \mathbb{R}^d$ of n points and $c > 1, D > 0$. Let $G = (P, E)$ be the output of Algorithm 1. Then $|E| \leq O(\mathbf{p}_1^{-1} n \log n)$. In addition, with success probability at least $1 - O(\max(1/n^3, \mathbf{p}_2 \mathbf{p}_1^{-1} n^2 \log n))$, the following holds: (1) $\forall p, q \in P$ with $\operatorname{dist}(p, q) \leq D$, either there is an edge between p, q in G or p, q have a common neighbor in G. (2) If there is an edge between $p, q \in P$ in G, $\operatorname{dist}(p, q) < cD$.*

Algorithm 1 Near Neighbor Graph via LSH

1: **Input:** n points $P \subset \mathbb{R}^d, c > 1, D > 0$.
2: Choose a $(D, cD, \mathbf{p}_1, \mathbf{p}_2)$-sensitive family $\mathcal{H}$ and draw independent $h_1, \cdots, h_t$ from $\mathcal{H}$ for $t = \lceil \frac{5 \log n}{\mathbf{p}_1} \rceil$.
3: Create a graph $G = (P, E)$ where each point $p \in P$ denotes a vertex in the graph.
4: For each hash function $h_i, i \in [t]$ and each point $p \in P$, connects p, q in G where $q \in P$ is the point with the smallest index such that $h_i(p) = h_i(q)$.
5: Return G.

3.3. Clustering Algorithm

3.3.1. Finding Candidate Clusters

Let $\alpha' = (\alpha/4 - 1/2)/1.01$. Without loss of generality, we assume that $\min_{p \neq q \in P} \operatorname{dist}(p, q) \geq \alpha'$. Otherwise we can scale P to make the closest distance at least α'. Let Δ be an upper bound of the distance between furthest points in P, i.e., $\Delta \geq \max_{p,q \in P} \operatorname{dist}(p, q)$. Suppose Δ is a power of 1.01 and $L := \log_{1.01} \Delta$. Algorithm 2 outputs the candidates of optimal clusters.

Algorithm 2 Candidates of Optimal Clusters

1: **Input:** $\alpha' > 1$, n points $P \subset \mathbb{R}^d$ with $\min_{p \neq q \in P} \operatorname{dist}(p, q) \geq \alpha'$ and $\max_{p,q} \operatorname{dist}(p, q) \leq \Delta$.
2: For $i \in \{0, 1, \cdots, L\}$ where $L = \log_{1.01} \Delta$, run Algorithm 1 with $P, c = \alpha', D = 1.01^i$, and let $G_i = (P, E_i)$ be the output.
3: For $i \in \{0, 1, \cdots, L\}$, let $G_i' = (P, E_0 \cup E_1 \cup \cdots \cup E_i)$.
4: For $i \in \{0, 1, \cdots, L\}$, compute an arbitary $\mathcal{Q}_i = \{Q_{i,1}, Q_{i,2}, \cdots, Q_{i,s_i}\}$ such that (1). each $Q_{i,j} \subseteq P$ is a connected component of G_i' (2). if a connected component C in G_i' has diameter at most 2, then $C \in \mathcal{Q}_i$.
5: Return $\mathcal{Q}_0, \mathcal{Q}_1, \cdots, \mathcal{Q}_L$.

Theorem 3.6 (Candidates of optimal clusters)**.** *Consider a point set P which satisfies α-center proximity property for k-means. Let $\alpha' = (\alpha/4 - 1/2)/1.01$ and let $\mathcal{Q}_0, \mathcal{Q}_1, \cdots, \mathcal{Q}_L$ be the output of Algorithm 2 for P, α'. Let $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$ be the optimal k-means clustering for P. Suppose the failure probability of Algorithm 1 is at most $\mathbf{p}_3$. With success probability at least $1 - O(\mathbf{p}_3 \log \Delta)$, $\forall C \in \mathcal{C}, C \in \mathcal{Q}_0 \cup \mathcal{Q}_1 \cup \cdots \cup \mathcal{Q}_L$, and $P \in \mathcal{Q}_L, \forall p \in P, \{p\} \in \mathcal{Q}_0$.*

Thus, the problem becomes to choose clusters $C_1, C_2, \cdots, C_k \in \mathcal{Q}_1 \cup \mathcal{Q}_2 \cup \cdots \cup \mathcal{Q}_L$ such that $C_1, C_2, \cdots, C_k$ is a partition of P and the clustering cost is minimized. We will show how to use dynamic programming to find the optimal clusters from $\mathcal{Q}_1 \cup \mathcal{Q}_2 \cup \cdots \cup \mathcal{Q}_L$.

3.3.2. Tree Construction over Candidate Clusters

We first build a binary tree $\mathcal{T}'$ over candidate clusters. Later we run a dynamic programming over $\mathcal{T}'$. In Algorithm 3, we show how to build a tree $\mathcal{T}$ over candidate clusters. In Algorithm 4, we show how to convert $\mathcal{T}$ to a binary tree $\mathcal{T}'$.

Algorithm 3 Tree Construction over Candidate Clusters

1: **Input:** $\mathcal{Q}_0, \mathcal{Q}_1, \cdots, \mathcal{Q}_L$ outputted by Algorithm 2.
2: For each $C \in \mathcal{Q}_0 \cup \mathcal{Q}_1 \cup \cdots \cup \mathcal{Q}_L$, create a vertex representing C in tree $\mathcal{T}$.
3: For $i \in \{0, 1, \cdots, L-1\}$ and each $C \in \mathcal{Q}_i$, find the smallest $i' > i$ such that $\exists C' \in \mathcal{Q}_{i'}, C \subseteq C'$, set the parent of C to be C' in $\mathcal{T}$.
4: Return $\mathcal{T}$.

Lemma 3.7 ($\mathcal{T}$ is a small-depth tree)**.** *Let $\mathcal{Q}_0, \cdots, Q_L$ be the output of Algorithm 2. Let $\mathcal{Q} = \bigcup_{i=0}^{L} \mathcal{Q}_i$. Suppose the failure probability of Algorithm 1 is at most $\mathbf{p}_3$. With probability at least $1 - O(\mathbf{p}_3)$, $P \in \mathcal{Q}_L$ and $\forall p \in P, \{p\} \in \mathcal{Q}_0$. Conditioned on $P \in \mathcal{Q}$, the output $\mathcal{T}$ of Algorithm 3 is a tree over $\mathcal{Q}$. Furthermore, the depth of $\mathcal{T}$ is at most L.*

Definition 3.8. A pruning set $\mathcal{S}$ of tree $\mathcal{T}$ is a subset of vertices of $\mathcal{T}$ such that for any leaf, there is a unique vertex $C \in \mathcal{S}$ on the path from the leaf to the root.

Lemma 3.9 (Properties of $\mathcal{T}$)**.** *Let $\mathcal{Q} = \bigcup_{i=0}^{L} \mathcal{Q}_i$. Let $\mathcal{T}$ be the output of Algorithm 3. Conditioned on $P \in \mathcal{Q}$ and $\forall p \in P, \{p\} \in \mathcal{Q}$, any pruning set of $\mathcal{T}$ is a partition of P, and any partition $\mathcal{C} = \{C_1, C_2, \cdots, C_s\}$ of P is a prunning set of $\mathcal{T}$ if $\mathcal{C}$ satisfies $\forall C \in \mathcal{C}, C \in \mathcal{Q}$*

In Algorithm 4, we show how to convert $\mathcal{T}$ obtained by Algorithm 3 to a binary tree.

Algorithm 4 Conversion to A Binary Tree

1: **Input:** $\mathcal{T}$ outputted by Algorithm 3.
2: Add all vertices of $\mathcal{T}$ into $\mathcal{T}'$.
3: Consider each non-leaf vertex C and its children $C_1, C_2, \cdots, C_s$ in $\mathcal{T}$. Create s auxiliary vertices $C'_1, C'_2, \cdots, C'_s$ in $\mathcal{T}'$. For $i \in [s]$, we set the parent of C_i in $\mathcal{T}'$ as $C'_{\lfloor (s+i)/2 \rfloor}$. For each $i \in \{2, 3, \cdots, s\}$, we set the parent of C'_i in $\mathcal{T}'$ as $C'_{\lfloor i/2 \rfloor}$. We set the parent of C'_1 in $\mathcal{T}'$ as C. Furthermore, for each $i \in [s]$, we use C'_i to represent the point set which is the union of point sets represented by its children.
4: Return $\mathcal{T}'$ and $\mathcal{Q}'$ where $\mathcal{Q}'$ is the vertex set of $\mathcal{T}'$.

Lemma 3.10 (Properties of $\mathcal{T}'$)**.** *Consider the output $\mathcal{T}'$ of Algorithm 4. Each vertex in $\mathcal{T}'$ has at most 2 children. The depth of $\mathcal{T}'$ is at most $\log n$ times larger than the depth of $\mathcal{T}$. Any pruning set of $\mathcal{T}$ is also a pruning set of $\mathcal{T}'$. In addition, conditioned on that any pruning set of $\mathcal{T}$ is a partition of P, any pruning set of $\mathcal{T}'$ is a partition of P.*

3.3.3. Dynamic Programming over the Tree

According to Theorem 3.6, Lemma 3.9 and Lemma 3.10, to find the optimal k-means clustering of a point set which satisfies α-center proximity property for k-means, we only need to find the optimal size-k pruning set of $\mathcal{T}'$. Recall that each vertex C of tree $\mathcal{T}'$ represents a subset of points of P, i.e., $C \subseteq P$. For each vertex C of $\mathcal{T}'$, we assign it a weight $w(C) = \sum_{p \in C} \mathrm{dist}^2(p, \mathrm{mean}(C))$. To solve k-means for P, our goal is to find a pruning set $\mathcal{S}$ of $\mathcal{T}'$ with $|\mathcal{S}| = k$ such that $\sum_{C \in \mathcal{S}} w(C)$ is minimized. In Algorithm 5, we show a dynamic programming algorithm which computes the optimal size-k pruning set of $\mathcal{T}'$.

Algorithm 5 Dynamic Programming for Exact Solution

1: **Input:** Tree $\mathcal{T}'$ where each vertex has at most 2 children, weights $w(\cdot)$ of vertices, and parameter k.
2: For each leaf vertex C of $\mathcal{T}'$, compute $\mathrm{DPcost}(C, i) = w(C)$ for all $i \in [k]$.
3: **for** $\exists$vertex C, $\mathrm{DPcost}(C, \cdot)$ are not computed yet **do**
4: Find a vertex C such that $\mathrm{DPcost}(C, \cdot)$ are not computed, but $\mathrm{DPcost}(C', i)$ is computed for every child C' of C and every $i \in [k]$.
5: If C only has one child C', for $i \in [k]$, compute $\mathrm{DPcost}(C, i) = \min(w(C), \mathrm{DPcost}(C', i))$.
6: If C has two children C', C'', compute $\mathrm{DPcost}(C, 1) = w(C)$, and for $i \in \{2, 3, \cdots, k\}$, compute $\mathrm{DPcost}(C, i) = \min(w(C), \min_{i' \in [i-1]} \mathrm{DPcost}(C', i') + \mathrm{DPcost}(C'', i - i'))$.
7: **end for**
8: //Traverse backwards to get the pruning set:
9: Initialize the pruning set $\mathcal{S} = \emptyset$, and a list $\mathcal{L} = \{(R, k)\}$ where R is the root of $\mathcal{T}'$.
10: Repeat the following until $\mathcal{L}$ is empty:
  1. Pick an arbitrary (C, i) from $\mathcal{L}$ and remove it from $\mathcal{L}$.
  2. If $\mathrm{DPcost}(C, i) = w(C)$, add C into $\mathcal{S}$ and skip following steps.
  3. If C only has one child C', add (C', i) into $\mathcal{L}$.
  4. If C has two children C', C'', find the smallest $i' \in [i-1]$ such that $\mathrm{DPcost}(C, i) = \mathrm{DPcost}(C', i') + \mathrm{DPcost}(C'', i - i')$, and add (C', i') and $(C'', i - i')$ into $\mathcal{L}$.
11: Output $\mathcal{S}$.

Lemma 3.11 (Optimal size-k pruning set)**.** *Algorithm 5 outputs a pruning set $\mathcal{S}$ of $\mathcal{T}'$ with $|\mathcal{S}| \le k$ such that $\sum_{C \in \mathcal{S}} w(C) = \min_{\text{pruning set } \mathcal{S}' \text{ of } \mathcal{T}' : |\mathcal{S}'| \le k} \sum_{C \in \mathcal{S}'} w(C)$.*

In Algorithm 6, we show an alternative dynamic programming algorithm which computes an approximately optimal size-k pruning set of $\mathcal{T}'$. In comparison with Algorithm 5, the space needed by Algorithm 6 is independent of k.

Lemma 3.12 (Approximately optimal size-k pruning set)**.** *Algorithm 6 outputs a pruning set $\mathcal{S}$ of $\mathcal{T}'$ with $|\mathcal{S}| \le k$ such that $\sum_{C \in \mathcal{S}} w(C) \le (1 + \varepsilon) \cdot \min_{\text{pruning set } \mathcal{S}' \text{ of } \mathcal{T}' : |\mathcal{S}'| \le k} \sum_{C \in \mathcal{S}'} w(C)$.*

By running Algorithm 2, Algorithm 3, Algorithm 4 and Algorithm 5 or Algorithm 6, we are able to find k-means or approximate k-means clustering. The correctness follows from Theorem 3.6, Lemma 3.9, Lemma 3.10, Lemma 3.11 and Lemma 3.12. We refer readers to Appendix B.9 for more discussions for guarantees of the offline algorithms.

4. Implementation in the MPC Model

In this section, we discuss the implementation of algorithms described in the previous section into the MPC model. We put all missing proofs and missing details in Appendix C.

4.1. Construction of Near Neighbor Graph

Many LSH families for Euclidean space can be easily implemented in the MPC model. The following is an Euclidean LSH with near optimal guarantee. We put the detailed MPC

Algorithm 6 Dynamic Programming for Approximate Solution

Input: Tree $\mathcal{T}'$ where each vertex has at most 2 children, weights $w(\cdot)$ of vertices, parameters $k \in [n], \varepsilon \in (0, 0.5)$. Weights are non-negative and the minimum positive weight $w(C)$ is at least 1. The maximum weight is $w(R)$ where R is the root of $\mathcal{T}'$. $w(C) = 0$ for each leaf vertex C in $\mathcal{T}'$.
Let $\varepsilon' = \varepsilon/(2H)$ where H is the depth of $\mathcal{T}'$. Let l be the smallest integer with $(1+\varepsilon')^l \geq 2n \cdot w(R)$.
Let $\lambda_{-1} = 0$ and $\lambda_i = (1+\varepsilon')^i$ for $i \in \{0, 1, \cdots, l\}$.
For each leaf vertex C of $\mathcal{T}'$, compute $\mathrm{DPsize}(C, i) = 1$ for $i \in \{-1, 0, 1, \cdots, l\}$.
for $\exists$vertex C, $\mathrm{DPsize}(C, \cdot)$ are not computed yet **do**
  Find a vertex C such that $\mathrm{DPsize}(C, \cdot)$ are not computed, but $\mathrm{DPsize}(C', i)$ is computed for every child C' of C and every $i \in \{-1, 0, \cdots, l\}$.
  Let $i' \in \{-1, 0, \cdots, l\}$ be the smallest value such that $\lambda_{i'} \geq w(C)$. Compute $\mathrm{DPsize}(C, i) = 1$ for $i \in \{i', i'+1, \cdots, l\}$.
  If C only has one child C', for $i \in \{-1, 0, \cdots, i'-1\}$, compute $\mathrm{DPsize}(C, i) = \mathrm{DPsize}(C', i)$.
  If C has children C', C'', compute $\mathrm{DPsize}(C, i) = \min_{i', i'' \in \{-1,0,\cdots,l\}: \lambda_{i'} + \lambda_{i''} \leq \lambda_i} \mathrm{DPsize}(C', i') + \mathrm{DPsize}(C'', i'')$
end for
//Traverse backwards to get the pruning set:
Find the smallest $i^* \in \{-1, 0, \cdots, l\}$ such that $\mathrm{DPsize}(R, i^*) \leq k$.
Initialize the pruning set $\mathcal{S} = \emptyset$, and a list $\mathcal{L} = \{(R, i^*)\}$.
Repeat the following until $\mathcal{L}$ is empty:
  1. Pick an arbitrary (C, i) from $\mathcal{L}$ and remove it from $\mathcal{L}$.
  2. If $\mathrm{DPsize}(C, i) = 1$, add C into $\mathcal{S}$ and skip following steps.
  3. If C only has one child C', add (C', i) into $\mathcal{L}$.
  4. If C has children C', C'', find (i', i'') with the smallest lexicographical order such that $\lambda_{i'} + \lambda_{i''} \leq \lambda_i$ and $\mathrm{DPsize}(C', i') + \mathrm{DPsize}(C'', i'') = \mathrm{DPsize}(C, i)$. Add (C', i') and (C'', i'') into $\mathcal{L}$.
Return $\mathcal{S}$.

implementation of it into Appendix A.

Lemma 4.1 ((Andoni & Indyk, 2006)). *For any "scale" $D > 0$, dimension $d > 0$, $c > 0$ and $s \in [1, \log n]$, there is a $(D, cD, 1/n^{s/c^2+o(1)}, 1/n^s)$-sensitive family $\mathcal{H}$ of hashing functions for $\mathbb{R}^d$. In addition, for any set of n points $P \subset \mathbb{R}^d$ and any $h \in \mathcal{H}$, there is an fully scalable MPC algorithm with $O(1)$ rounds and $n^{1+o(1)}d$ total space that computes $h(p)$ for every $p \in P$. The space to store each $h(p)$ is $O(\log^3 n)$.*

By using above LSH construction, we can implement Algorithm 1 by a fully scalable MPC algorithm. Notice that the edges of the output graph can be distributed arbitrarily over machines. We can always use sorting (see Theorem 2.2) to reorder the edges on the machines.

Lemma 4.2 (Algorithm 1 in MPC). *Suppose $c \geq 1 + 10/\log n$. Algorithm 1 can be implemented in the MPC model in $O(1)$ rounds using $n^{1+5/(c^2-1)+o(1)}d$ total space. Furthermore, the algorithm is fully scalable and the success probability is at least $1 - O(1/n^2)$. The size of the output graph G is at most $n^{1+5/(c^2-1)+o(1)}$.*

4.2. Finding Candidate Clusters in MPC

We can use Lemma 4.2 as a building block to implement Algorithm 2 in the MPC model. Suppose we know Δ for the point set P. In Appendix C.7, we will show how to get a good estimation of Δ. Recall that $L = O(\log \Delta)$. We mildly assume that $\log \Delta = n^{o(1)}$. We use the representation method mentioned in (Andoni et al., 2018) to store sets in the MPC model. In particular, for each set C stored in the system, C is not necessarily to be stored on a single machine, i.e., C can be stored in a distributed manner over machines. For each element $p \in C$, we only need to store a pair ("C", p) in an arbitrary machine, where "C" is the name of C. It is easy to use sorting (see Theorem 2.2) to implement set operations such as removing duplicates, redistributing the elements over machines, computing the set size, etc. We refer readers to (Andoni et al., 2018) for more MPC set operations. Consider $\mathcal{Q}$ which is a family of sets. Similarly, we do not need to store entire $\mathcal{Q}$ on a single machine. For each set $C \in \mathcal{Q}$, we only need to store a pair ("$\mathcal{Q}$", "C") on an arbitrary machine.

Lemma 4.3 (Algorithm 2 in MPC). *Algorithm 2 can be implemented in the MPC model in $O(1)$ rounds and $n^{1+5/(\alpha'^2-1)+o(1)}d$ total space, i.e., all $\mathcal{Q}_0, \cdots, \mathcal{Q}_L$ and all $C \in \mathcal{Q}_0 \cup \cdots \mathcal{Q}_L$ are stored in the system at the end of the algorithm. Furthermore, the algorithm is fully scalable and the success probability is at least $1 - O(1/n^{1.99})$.*

4.3. MPC Construction of Tree over Candidate Clusters

In this section, we show how to implement Algorithm 3 and Algorithm 4 in the MPC model. In addition to compute the traditional representation of a rooted tree — parent pointers of vertices (Andoni et al., 2018; 2019), we also compute every leaf-to-root path and store each path on a single machine. According to Lemma 3.7 and Lemma 3.10, the depth of $\mathcal{T}$ is $O(L)$ and the depth of $\mathcal{T}'$ is $O(L \log n)$. Thus both depths are $n^{o(1)}$. Each leaf-to-root path has length at most $n^{o(1)}$ and can be fitted in the local memory of a single machine.

Lemma 4.4 (Algorithm 3 in MPC). *Conditioned on that Algorithm 2 succeeds, Algorithm 3 can be implemented in the MPC model in $O(1)$ rounds and $n^{1+o(1)}$ total space. The algorithm is fully scalable. The algorithm computes the parent of each vertex in $\mathcal{T}$ and computes all paths from leafs to the root. Each leaf-to-root path is stored on a single machine.*

Similar to Lemma 4.4, we show the MPC implementation of Algorithm 4.

Lemma 4.5. *Suppose the parent of each vertex in $\mathcal{T}$ is computed and each leaf-to-root path of $\mathcal{T}$ is stored on a single machine after running Algorithm 3. Algorithm 4 can be implemented in the MPC model in $O(1)$ rounds and $n^{1+o(1)}$ total space. The algorithm is fully scalable. The*

algorithm computes the parent of each vertex in $\mathcal{T}'$ and computes all paths from leafs to the root. Each leaf-to-root path is stored on a single machine. Furthermore, for each vertex C in $\mathcal{T}'$, $w(C) = \sum_{p\in C} \mathrm{dist}^2(p, \mathrm{mean}(C))$ is also computed.

4.4. An MPC Framework for Dynamic Programming over A Tree

We show how to implement Algorithm 5 and Algorithm 6 by the same framework in MPC (Algorithm 7). The high level idea is that since we have all leaf-to-root path, we know all ancestors of each vertex. Furthermore, since the depth of the tree is small, the total size over all subtrees is small. Thus, we are able to generate all subtrees in $O(1)$ rounds with small total space. Then, we are able to send each small subtree to a single machine and handle it locally.

Algorithm 7 Framework for Dynamic Programming in MPC

1: **Input:** $\mathcal{T}'$ with weights $w(\cdot)$ of vertices.
2: **Setup:** Let $\mathrm{DP}(C,\cdot)$ denote either $\mathrm{DPcost}(C,\cdot)$ in Algorithm 5 or $\mathrm{DPsize}(C,\cdot)$ in Algorithm 6. Let B be an upper bound of the space needed to store all $\mathrm{DP}(C,i)$ for all valid i of any vertex C. Each machine has local memory $\Theta(n^{\delta'}\cdot B)$ for some constant $\delta' > 0$.
3: Mark each vertex C in $\mathcal{T}'$ as *ongoing*.
4: **for** ongoing vertex exists **do**
5: For each vertex C in $\mathcal{T}'$ compute $\mathcal{S}_C = \{C' \mid C'$ is ongoing and C' is in the subtree rooted at $C\}$.
6: For each vertex C in $\mathcal{T}'$, if $|\mathcal{S}_C| \le n^{\delta'}$, send $\mathcal{S}_C$ to a single machine and compute $\mathrm{DP}(C', i)$ for every $C' \in \mathcal{S}_C$ and every valid i.
7: For each vertex C in $\mathcal{T}'$, if $\mathrm{DP}(C,\cdot)$ are computed by any machine, mark C as *finished*.
8: **end for**
9: For each leaf-to-root path $C_1, C_2, \cdots, C_m$ in $\mathcal{T}'$, based on $w(C_i), w(C_{i+1}), \cdots, w(C_m)$, $\mathrm{DP}(C_i,\cdot), \mathrm{DP}(C_{i+1},\cdot), \cdots, \mathrm{DP}(C_m,\cdot)$ and $\mathrm{DP}(C'_i,\cdot), \mathrm{DP}(C'_{i+1},\cdot), \cdots, \mathrm{DP}(C'_{m-1},\cdot)$ where C'_j (if exists) is a child of C_{j+1} other than C_j, we determine whether C_i should be added into the output $\mathcal{S}$ (of Algorithm 5 or Algorithm 6).
10: Remove duplicates in $\mathcal{S}$ and output $\mathcal{S}$.

Lemma 4.6. *Suppose each leaf-to-root path of $\mathcal{T}'$ is stored on a single machine and $w(C)$ for each vertex C in $\mathcal{T}'$ is computed after running Algorithm 4. Let B be defined in Algorithm 7. Algorithm 7 can be implemented in the MPC model using $O(1)$ rounds and total space $B\cdot n^{1+o(1)}$. The local memory required per machine is $\Theta(n^{\delta'}\cdot B)$, where $\delta' \in (0,1)$ is an arbitrary constant. For $\mathrm{DP}(\cdot,\cdot) \equiv \mathrm{DPcost}(\cdot,\cdot)$, the output $\mathcal{S}$ is the same as Algorithm 5. For $\mathrm{DP}(\cdot,\cdot) \equiv \mathrm{DPsize}(\cdot,\cdot)$, the output $\mathcal{S}$ is the same as Algorithm 6.*

4.5. Final MPC k-Means Algorithms

Theorem 4.7 (Exact k-means)**.** *Let $\alpha > 6.05$. Let $\delta \in (0,1)$ be an arbitrary constant. Consider a set $P \subset \mathbb{R}^d$ of n points which is α-perturbation resilient to k-means and the aspect ratio of P is at most $2^{n^{o(1)}}$. There is an MPC algorithm which outputs the optimal k-means clustering of P with probability at least $1 - 1/\mathrm{poly}(n)$ using $O(1)$ rounds and $n^{1+o(1)}(k + n^{O(1/\alpha^2)}d)$ total space. The memory needed per machine is $\Theta(n^\delta k)$.*

Proof. According to Lemma 3.2, P satisfies α-center proximity for k-means. Let $\alpha' = (\alpha/4 - 1/2)/1.01$. Then $\alpha' > 1.002$. Let $\mathcal{C}^* = \{C_1^*, C_2^*, \cdots, C_k^*\}$ be the optimal k-means clustering for P. Consider the MPC model that each machine has local memory $\Theta(n^\delta k)$. According to Lemma 4.3, we can run Algorithm 2 in the MPC model using $O(1)$ rounds and $n^{1+O(1/\alpha^2)+o(1)}d$ total space. The success probability is at least $1 - 1/\mathrm{poly}(n)$. Let $\mathcal{Q}_0, \mathcal{Q}_1, \cdots, \mathcal{Q}_L$ be the output of Algorithm 2. In the remaining, we condition on that Algorithm 2 succeeds. According to Theorem 3.6, $\forall i \in [k], C_i^* \in \mathcal{Q}_0 \cup \mathcal{Q}_1 \cup \cdots \cup \mathcal{Q}_L$, and $P \in \mathcal{Q}_L, \forall p \in P, \{p\} \in \mathcal{Q}_0$. Then according to Lemma 4.4, we can run Algorithm 3 in the MPC model using $O(1)$ rounds and $n^{1+o(1)}$ total space. Let $\mathcal{T}$ be the output of Algorithm 3. According to Lemma 3.9, $\mathcal{C}^*$ is a pruning set of $\mathcal{T}$ and for any pruning set $\mathcal{S}$ of $\mathcal{T}$ with $|\mathcal{S}| \le k$, we have $\sum_{C\in\mathcal{C}^*}\sum_{p\in C}\mathrm{dist}^2(p,\mathrm{mean}(C)) \le \sum_{C\in\mathcal{S}}\sum_{p\in C}\mathrm{dist}^2(p,\mathrm{mean}(C))$. According to Lemma 4.5, we can run Algorithm 4 in $O(1)$ rounds and $n^{1+o(1)}$ total space. Let $\mathcal{T}'$ be the output of Algorithm 4. For each C in $\mathcal{T}'$, $w(C) = \sum_{p\in C}\mathrm{dist}^2(p,\mathrm{mean}(C))$ is also computed. According to Lemma 3.10, $\mathcal{C}^*$ is a pruning set of $\mathcal{T}'$ and for any pruning set $\mathcal{S}$ of $\mathcal{T}'$ with $|\mathcal{S}| \le k$, we have $\sum_{C\in\mathcal{C}^*} w(C) \le \sum_{C\in\mathcal{S}} w(C)$. Since we need space $\Theta(k)$ to store $\mathrm{DPcost}(C,1), \mathrm{DPcost}(C,2), \cdots, \mathrm{DPcost}(C,k)$ for each C, according to Lemma 4.6, we can use Algorithm 7 to simulate Algorithm 5 in the MPC model using $O(1)$ rounds and $n^{1+o(1)}k$ total space. According to Lemma 3.11, we get a pruning set $\mathcal{S}$ of $\mathcal{T}'$ with $|\mathcal{S}| \le k$ such that $\sum_{C\in\mathcal{S}} w(C) = \sum_{C\in\mathcal{C}^*} w(C)$. According to Lemma 3.10, $\mathcal{S}$ is a partition of P and thus $\mathcal{S}$ is the optimal k-means clustering of P. The overall parallel time is $O(1)$. The total space needed is at most $n^{1+O(1/\alpha^2)+o(1)}d + n^{1+o(1)}k$. □

Theorem 4.8 (Approximate k-means)**.** *Let $\alpha > 6.05$ and $\varepsilon \in (1/n^{o(1)}, 0.5)$. Consider a set $P \subset \mathbb{R}^d$ of n points which is α-perturbation resilient to k-means and the aspect ratio of P is at most $2^{n^{o(1)}}$. There is a fully scalable MPC algorithm which outputs the $(1+\varepsilon)$-approximate k-means clustering of P with probability at least $1 - 1/\mathrm{poly}(n)$ using $O(1)$ rounds and $n^{1+O(1/\alpha^2)+o(1)}d$ total space.*

In Appendix C.7, we show how to reduce the dependence on d of Theorem 4.7 and Theorem 4.8 to finally get Theorem 1.1 and Theorem 1.2.

5. Experiments

In addition to the formal analysis of the theoretical guarantees of our k-means algorithms, we also conduct preliminary

experiments on both synthetic datasets and real datasets in the offline setting to further investigate their practical performances. All codes are in C++. We ran experiments on a machine with 16G RAM and Intel Core i7-3720QM@2.60GHz CPU. All experiments were in single threaded mode.

Chosen parameters and LSH. Instead of using the theoretically optimal LSH presented in Lemma 4.1, we implement the LSH of (Datar et al., 2004), which is simpler and also has theoretical guarantee. The hash function for the scale D is defined by a parameter $r \in \mathbb{R}_{>0}$, a set of m standard random Gaussian vectors $v_1, v_2, \cdots, v_m \in \mathbb{R}^d$ and a set of random variables $b_1, b_2, \cdots, b_m \in \mathbb{R}$ drawn from $[0, r]$ uniformly at random. The hash value $h(x)$ for a point $x \in \mathbb{R}^d$ is computed as $\left(\lfloor \frac{x^\top v_1/D+b_1}{r} \rfloor, \lfloor \frac{x^\top v_2/D+b_2}{r} \rfloor, \cdots, \lfloor \frac{x^\top v_m/D+b_m}{r} \rfloor\right)$. We use the same choice as (Datar et al., 2004), i.e., $r = 4$ and $m = 10$, and thus we choose t of Algorithm 1 to be 100. For Algorithm 2, instead of running Algorithm 1 for each scale 1.01^i, we run for each scale 1.1^i. For Algorithm 6, we choose $\varepsilon' = 0.1$.

5.1. Experiments on Synthetic Datasets

We generate synthetic dataset in the following way. We firstly choose k centers. Each center is drawn uniformly at random from $[0, 50]^d$. We create equal-size clusters for centers such that the total number of points is $n = 10^5$. The points of a cluster are drawn from the standard Gaussian distribution at the corresponding center. We denote this clustering as $\mathcal{C}^*$.

k-Means cost on synthetic datasets. We obtain the clustering $\mathcal{C}$ via Algorithm 5 or Algorithm 6. We calculate the ratio $\rho = \text{cost}(\mathcal{C})/\text{cost}(\mathcal{C}^*)$ by repeating 30 runs and taking the average. Condition on that $\mathcal{C}^*$ is the optimal k-means clustering, ρ should be at least 1. We test our algorithms on 10 groups of dataset, the parameters of the datasets are $n = 10^5, d \in \{10, 100\}, k \in \{5, 10, 25, 100, 1000\}$.

Due to the construction of synthetic datasets, they are well-perturbation-resilient. For all 10 synthetic datasets, our algorithms always achieve $\rho = 1.00(\pm 0.00)$, i.e., the optimal k-means clustering is obtained. Surprisingly, our approximate k-means algorithm (Algorithm 6) also outputs optimal clusters for these datasets.

Running time comparison between Algorithm 5 and Algorithm 6. We run both algorithms on the dataset with $n = 10^5$ and $d = 100$, and we evaluate the running time for various $k \in [50, 1000]$. As shown in figure 1, when k is small, Algorithm 5 is faster. But the running time of Algorithm 5 grows quickly in k while the running time of Algortihm 6 does not change. When k is sufficiently large, Algortihm 6 is faster.

5.2. Experiments on Real Datasets

Note that the k-means cost of Algorithm 5 cannot be worse than Algorithm 6 by our analysis. In this section, we evaluate the k-means cost of Algorithm 6 on real datasets listed in Table 1, and we compare it with the results obtained by k-means solver using k-means++ seeding implemeted by Python scikit-learn package (Pedregosa et al., 2011).

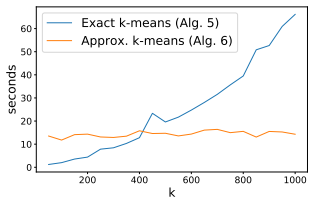

Figure 1. We run both Algorithm 5 and Algorithm 6 to compute k-means for the dataset with $n = 10^5, d = 100$. Algorithm 5 is faster when k is small and Algorithm 6 is faster when k is large.

Table 1. Real datasets. n: number of points, d: dimension of points , k: number of classes (clusters). All datasets can be obtained by sklearn.datasets.fetch_openml of the Python scikit-learn package.

DATASET	n	d	k
EUCA	736	14	5
MICE	1080	77	8
BIOMED	209	7	2
DIABETES	768	8	2
MAGIC	19020	10	2
GESTURE	9873	32	5
MNIST	60000	784	10
ZOO	101	16	7

Notice that k-means++ is a good sequential algorithm and should have a lower cost. Our goal is to show the cost by our algorithm is within a small factor of the cost by k-means++.

k-Means cost on real datasets. We obtain the clustering $\mathcal{C}$ via Algorithm 6 and the clustering $\mathcal{C}^*$ via k-means++. We calculate the ratio $\rho = \text{cost}(\mathcal{C})/\text{cost}(\mathcal{C}^*)$ by repeating 30 runs and taking the average. The results are in Table 2.

Table 2. k-Means cost on real datasets. $\text{cost}(\mathcal{C})$: the cost obtained by Algorithm 6, $\text{cost}(\mathcal{C})$: the cost obtained by k-means++, ρ : ratio between $\text{cost}(\mathcal{C})$ and $\text{cost}(\mathcal{C}^*)$.

DATASET	$\text{cost}(\mathcal{C}^*)$	$\text{cost}(\mathcal{C})$	ρ
EUCA	5050.98	6020.51	1.19
MICE	49447.74	74782.30	1.51
BIOMED	1127.79	1342.03	1.19
DIABETES	5134.04	6107.07	1.19
MAGIC	136926.74	189910.43	1.39
GESTURE	253642.04	306541.12	1.21
MNIST	8.041E10	1.321E11	1.64
ZOO	125.56	204.31	1.62

By checking the distances from input points to centers obtained by k-means++, these real datasets only hold at most 1.12-center proximity. Thus, they are not well-perturbation-resilient. But even when the assumption of perturbation-resilience does not hold for these datasets, our algorithm still achieves reasonable k-means cost (within at most ~ 1.6 factor of the cost obtained by k-means++). It would be an interesting future direction to explore what affects the practical accuracy of our algorithms.

References

Agarwal, P. K., Har-Peled, S., and Varadarajan, K. R. Approximating extent measures of points. *Journal of the ACM (JACM)*, 51(4):606–635, 2004.

Andoni, A. *Nearest neighbor search: the old, the new, and the impossible*. PhD thesis, Massachusetts Institute of Technology, 2009.

Andoni, A. and Indyk, P. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *2006 47th annual IEEE symposium on foundations of computer science (FOCS'06)*, pp. 459–468. IEEE, 2006.

Andoni, A., Song, Z., Stein, C., Wang, Z., and Zhong, P. Parallel graph connectivity in log diameter rounds. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 674–685. IEEE, 2018.

Andoni, A., Stein, C., and Zhong, P. Log diameter rounds algorithms for 2-vertex and 2-edge connectivity. *arXiv preprint arXiv:1905.00850*, 2019.

Angelidakis, H., Makarychev, K., and Makarychev, Y. Algorithms for stable and perturbation-resilient problems. In Hatami, H., McKenzie, P., and King, V. (eds.), *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pp. 438–451. ACM, 2017. doi: 10.1145/3055399.3055487. URL https://doi.org/10.1145/3055399.3055487.

Arora, S., Raghavan, P., and Rao, S. Approximation schemes for euclidean k-medians and related problems. In Vitter, J. S. (ed.), *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pp. 106–113. ACM, 1998. doi: 10.1145/276698.276718. URL https://doi.org/10.1145/276698.276718.

Arthur, D. and Vassilvitskii, S. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1027–1035. Society for Industrial and Applied Mathematics, 2007.

Awasthi, P., Blum, A., and Sheffet, O. Center-based clustering under perturbation stability. *Information Processing Letters*, 112(1-2):49–54, 2012.

Bachem, O., Lucic, M., and Krause, A. Coresets for nonparametric estimation - the case of dp-means. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 209–217, 2015. URL http://jmlr.org/proceedings/papers/v37/bachem15.html.

Bachem, O., Lucic, M., Hassani, S. H., and Krause, A. Approximate k-means++ in sublinear time. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pp. 1459–1467, 2016. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12147.

Bachem, O., Lucic, M., and Krause, A. Scalable k - means clustering via lightweight coresets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pp. 1119–1127, 2018. doi: 10.1145/3219819.3219973. URL https://doi.org/10.1145/3219819.3219973.

Bahmani, B., Moseley, B., Vattani, A., Kumar, R., and Vassilvitskii, S. Scalable k-means++. *Proceedings of the VLDB Endowment*, 5(7):622–633, 2012.

Balcan, M. F. and Liang, Y. Clustering under perturbation resilience. *SIAM Journal on Computing*, 45(1):102–155, 2016.

Balcan, M.-F. F., Ehrlich, S., and Liang, Y. Distributed k-means and k-median clustering on general topologies. In *Advances in Neural Information Processing Systems*, pp. 1995–2003, 2013.

Bateni, M., Behnezhad, S., Derakhshan, M., Hajiaghayi, M., and Mirrokni, V. Massively parallel dynamic programming on trees. *arXiv preprint arXiv:1809.03685*, 2018.

Bateni, M., Esfandiari, H., Fischer, M., and Mirrokni, V. Extreme k-center clustering. In *AAAI*, pp. 273–284, 2021.

Beame, P., Koutris, P., and Suciu, D. Communication steps for parallel query processing. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pp. 273–284. ACM, 2013.

Beame, P., Koutris, P., and Suciu, D. Communication steps for parallel query processing. *Journal of the ACM (JACM)*, 64(6):1–58, 2017.

Bhaskara, A. and Wijewardena, M. Distributed clustering via lsh based data partitioning. In *International Conference on Machine Learning*, pp. 569–578, 2018a.

Bhaskara, A. and Wijewardena, M. Distributed clustering via LSH based data partitioning. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 570–579. PMLR, 10–15 Jul 2018b. URL http://proceedings.mlr.press/v80/bhaskara18a.html.

Byrka, J., Pensyl, T., Rybicki, B., Srinivasan, A., and Trinh, K. An improved approximation for k-median, and positive correlation in budgeted optimization. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pp. 737–756. SIAM, 2014.

Ceccarello, M., Pietracaprina, A., and Pucci, G. Solving k-center Clustering (with Outliers) in MapReduce and Streaming, almost as Accurately as Sequentially. *CoRR*, abs/1802.09205, 2018.

Cohen-Addad, V. A fast approximation scheme for low-dimensional k-means. In Czumaj, A. (ed.), *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pp. 430–440. SIAM, 2018. doi: 10.1137/1.9781611975031.29. URL https://doi.org/10.1137/1.9781611975031.29.

Cohen-Addad, V. and Schwiegelshohn, C. On the local structure of stable clustering instances. In Umans, C. (ed.), *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pp. 49–60. IEEE Computer Society, 2017. doi: 10.1109/FOCS.2017.14. URL https://doi.org/10.1109/FOCS.2017.14.

Cohen-Addad, V., Feldmann, A. E., and Saulpic, D. Near-linear time approximations schemes for clustering in doubling metrics. In Zuckerman, D. (ed.), *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pp. 540–559. IEEE Computer Society, 2019a. doi: 10.1109/FOCS.2019.00041. URL https://doi.org/10.1109/FOCS.2019.00041.

Cohen-Addad, V., Klein, P. N., and Mathieu, C. Local search yields approximation schemes for k-means and k-median in euclidean and minor-free metrics. *SIAM J. Comput.*, 48(2):644–667, 2019b. doi: 10.1137/17M112717X. URL https://doi.org/10.1137/17M112717X.

Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pp. 253–262, 2004.

Dean, J. and Ghemawat, S. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

Epasto, A., Mirrokni, V., and Zadimoghaddam, M. Scalable diversity maximization via small-size composable core-sets. In *31st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2019.

Feldman, D. and Langberg, M. A unified framework for approximating and clustering data. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pp. 569–578, 2011.

Ghaffari, M., Kuhn, F., and Uitto, J. Conditional hardness results for massively parallel computation from distributed lower bounds. In Zuckerman, D. (ed.), *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pp. 1650–1663. IEEE Computer Society, 2019. doi: 10.1109/FOCS.2019.00097. URL https://doi.org/10.1109/FOCS.2019.00097.

Gibbons, A. and Rytter, W. *Efficient parallel algorithms*. Cambridge University Press, 1989.

Goodrich, M. T. Communication-efficient parallel sorting. *SIAM Journal on Computing*, 29(2):416–432, 1999.

Goodrich, M. T., Sitchinava, N., and Zhang, Q. Sorting, searching, and simulation in the mapreduce framework. In *International Symposium on Algorithms and Computation*, pp. 374–383. Springer, 2011.

Jain, K., Mahdian, M., Markakis, E., Saberi, A., and Vazirani, V. V. Greedy facility location algorithms analyzed using dual fitting with factor-revealing lp. *Journal of the ACM (JACM)*, 50(6):795–824, 2003.

Johnson, W. B. and Lindenstrauss, J. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., and Wu, A. Y. A local search approximation algorithm for k-means clustering. *Computational Geometry*, 28(2-3):89–112, 2004.

Karloff, H., Suri, S., and Vassilvitskii, S. A model of computation for mapreduce. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pp. 938–948. SIAM, 2010.

Kolliopoulos, S. G. and Rao, S. A nearly linear-time approximation scheme for the euclidean k-median problem. *SIAM J. Comput.*, 37(3):757–782, 2007. doi: 10.1137/S0097539702404055. URL https://doi.org/10.1137/S0097539702404055.

Li, S. A 1.488 approximation algorithm for the uncapacitated facility location problem. In *International Colloquium on Automata, Languages, and Programming*, pp. 77–88. Springer, 2011.

Lloyd, S. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

Lucic, M., Bachem, O., and Krause, A. Strong coresets for hard and soft bregman clustering with applications to exponential family mixtures. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016*, pp. 1–9, 2016. URL http://jmlr.org/proceedings/papers/v51/lucic16.html.

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

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Voevodski, K. Large scale k-median clustering for stable clustering instances. In *International Conference on Artificial Intelligence and Statistics*, pp. 2890–2898. PMLR, 2021.

White, T. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

A. Euclidean LSH in MPC

In this section, we show the construction of the LSH given by (Andoni & Indyk, 2006). We define $G^{t,w}$ to be the grid points in t-dimensional space where the side length of each grid cell is $4w$, i.e., $G^{t,w} = 4w \cdot \mathbb{Z}^t$. The Euclidean LSH (Andoni & Indyk, 2006) is given in Algorithm 8.

Algorithm 8 LSH for Points in Euclidean Spaces

1: **Preprocessing:**
2: Let $U \geq 1$. For $u \in [U]$, draw $v_u \in [0, 4w]^t$ uniformly at random. Let $G_u^{t,w} = G^{t,w} + v_u$ denote the points on the grid shifted by v_u.
3: Let $A \in \mathbb{R}^{t \times d}$ be a random matrix where each entry is drawn independently from $\mathcal{N}(0, 1/t)$.
4: **Evaluating $h(p)$ for $p \in \mathbb{R}^d$:**
5: Compute $p' = A \cdot p$. Find minimum $u \in [U]$ such that $\exists x \in G_u^{t,w}, \|p' - x\|_2 \leq w$. Set $h(p) = (u, x)$.

The following two lemmas for Algorithm 8 is shown by (Andoni, 2009).

The following lemma shows that if U is sufficiently large, then the balls over grids $G_1^{t,w}, G_2^{t,w}, \cdots, G_U^{t,w}$ cover the entire space $\mathbb{R}^t$ with high probability. Thus, $h(p)$ is well-defined for every $p \in \mathbb{R}^d$ with high probability.

Lemma A.1 (Lemma 3.2.2 of (Andoni, 2009))**.** *If $U = t^{\Theta(t)} \log n$, with probability at least $1 - 1/n^{10}$, $\forall p' \in \mathbb{R}^t, \exists u \in [U]$ s.t. $\exists x \in G_u^{t,w}, \|p' - x\|_2 \leq w$.*

Without loss of generality, we can only consider the scale $D = 1$. For different scale, we just need to multiply the corresponding scaling factor for the input points.

Lemma A.2 (Lemma 3.2.3 of (Andoni, 2009))**.** *Consider points $p, q \in \mathbb{R}^d$ and $c > 1$. Let P_1 be $\Pr_h[h(p) = h(q)]$ when $\|p - q\|_2 \leq 1$. Let P_2 be $\Pr_h[h(p) = h(q)]$ when $\|p - q\|_2 \geq c$. Suppose $w = t^{1/3}$. Then, (1). $\rho = \frac{\log 1/P_1}{\log 1/P_2} = 1/c^2 + O(1/t^{1/4})$. (2). $0.99 \geq P_2 \geq e^{-O(t^{1/3})}$.*

In particular, when $t = \log^{4/5} n$, we have $\rho = 1/c^2 + O(1/\log^{1/5} n)$. Let $s \in [1, \log n]$. Let us define $g(p) = (h_1(p), h_2(p), \cdots, h_b(p))$, where $h_1(\cdot), h_2(\cdot), \cdots, h_b(\cdot)$ are b independent hash functions described by Algorithm 8, and b is the smallest integer such that $P_2^b \leq 1/n^s$. Since $P_2 \leq 0.99$, we have $b \leq O(s \log n) = O(\log^2 n)$. The size of $g(p)$ is $(t+1) \cdot b \leq O(\log^3 n)$. Since $P_1 \geq P_2 \geq e^{-O(\log^{4/15} n)}$, we have $P_1^b \geq P_2^{\rho b} \geq 1/n^{s\rho + o(1)}$. Theorefore, $g(\cdot)$ has the following properties: (1). For any two points $p, q \in \mathbb{R}^d$ with $\|p - q\|_2 \leq 1$, $\Pr[g(p) = g(q)] \geq 1/n^{s\rho+o(1)} = 1/n^{s/c^2+o(1)}$. (2). For any two points $p, q \in \mathbb{R}^d$ with $\|p - q\|_2 \geq c$, $\Pr[g(p) = g(q)] \leq 1/n^s$. Thus, the hash family $\mathcal{H}$ of $g(\cdot)$ is $(D, cD, 1/n^{s/c^2+o(1)}, 1/n^s)$-sensitive for $\mathbb{R}^d$.

Lemma A.3. *Consider a set $P \subset \mathbb{R}^d$ of n points. There is a fully scalable MPC algorithm which computes $g(p)$ for every point $p \in P$ in $O(1)$ rounds. The total space needed is $n^{1+o(1)}d$.*

Proof. Notice that $g(p) = (h_1(p), h_2(p), \cdots, h_b(p))$ and $h_1, h_2, \cdots, h_b$ are independent hash functions described by Algorithm 8. We can compute these $b = O(\log^2 n)$ values $h_1(p), h_2(p), \cdots, h_b(p)$ simutaneously. It suffices to show how to apply one function $h(\cdot)$ for every point $p \in P$.

Let $t = \log^{4/5} n$. Consider the preprocessing step, according to Lemma A.1, U is at most $t^{\Theta(t)} \log n = n^{o(1)}$. We use one machine to generate shift vectors $v_1, v_2, \cdots, v_U \in \mathbb{R}^t$. We can use $O(1)$ rounds (Goodrich et al., 2011) to make every machine know $v_1, v_2, \cdots, v_U$. We can use multiple machines (in the case $t \cdot d$ is larger than the local memory) together generate the random Gaussian matrix $A \in \mathbb{R}^{t \times d}$ where each machine holds a part of entries of A. Notice that $A \cdot p$ can be computed in the COMBINING CRCW PRAM in $O(1)$ parallel time using $t \cdot d$ processors. By simulating this PRAM algorithm, we can compute $p' = A \cdot p$ for every $p \in P$ in $O(1)$ rounds. The algorithm is fully scalable and the total space needed is $ntd = n^{1+o(1)}d$. Since p' has size $t = n^{o(1)}$, p' can be held by a single machine. Since every machine knows $v_1, v_2, \cdots, v_U$, $h(p)$ can be computed locally. Thus, the parallel time needed is $O(1)$.

Since we simultaneously run b copies of the above procedure, the parallel time is still $O(1)$ and the total space needed is $ntd \cdot b = n^{1+o(1)}d$. □

B. Missing Details of Section 3

B.1. Proof of Lemma 3.3

Before we prove Lemma 3.3, we need to show the following lemma:

Lemma B.1. *Consider a point set P which satisfies α-center proximity property for k-means. Let $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$ be the optimal k-means clustering for P. $\forall C_i \neq C_j \in \mathcal{C}$ and $\forall p, q \in C_i, s \in C_j$, $(\alpha/2 - 1)\operatorname{dist}(p, q) < \min(\operatorname{dist}(p, s), \operatorname{dist}(q, s))$.*

Proof. Let $c_i = \operatorname{mean}(C_i)$, $c_j = \operatorname{mean}(C_j)$. If $\operatorname{dist}(p, c_i) < \operatorname{dist}(s, c_j)$, then we have $\alpha \operatorname{dist}(p, c_i) < \alpha \operatorname{dist}(s, c_j) < \operatorname{dist}(s, c_i) \leq \operatorname{dist}(s, p) + \operatorname{dist}(p, c_i)$ and thus $\operatorname{dist}(s, p) > (\alpha - 1)\operatorname{dist}(p, c_i)$. Otherwise $\operatorname{dist}(p, c_i) \geq \operatorname{dist}(s, c_j)$, we have $\alpha \operatorname{dist}(p, c_i) < \operatorname{dist}(p, c_j) \leq \operatorname{dist}(p, s) + \operatorname{dist}(s, c_j) \leq \operatorname{dist}(p, s) + \operatorname{dist}(p, c_i)$ and thus $\operatorname{dist}(s, p) > (\alpha - 1)\operatorname{dist}(p, c_i)$. Therefore, in either case, we have $\operatorname{dist}(s, p) > (\alpha - 1)\operatorname{dist}(p, c_i)$. By the same argument, we can also show $\operatorname{dist}(s, q) > (\alpha - 1)\operatorname{dist}(q, c_i)$.

Thus, we have $(\alpha-1)\operatorname{dist}(p, q) \leq (\alpha-1)(\operatorname{dist}(p, c_i)+\operatorname{dist}(c_i, q)) < \operatorname{dist}(s, p)+\operatorname{dist}(s, q)$. Since $\operatorname{dist}(s, p) \leq \operatorname{dist}(s, q)+\operatorname{dist}(p, q)$ and $\operatorname{dist}(s, q) \leq \operatorname{dist}(s, p)+\operatorname{dist}(p, q)$, we have $\operatorname{dist}(s, p)+\operatorname{dist}(s, q) \leq 2\min(\operatorname{dist}(s, p), \operatorname{dist}(s, q))+\operatorname{dist}(p, q)$. Therefore, $(\alpha - 1)\operatorname{dist}(p, q) < 2\min(\operatorname{dist}(p, s), \operatorname{dist}(q, s)) + \operatorname{dist}(p, q)$ which implies that $(\alpha/2 - 1)\operatorname{dist}(p, q) < \min(\operatorname{dist}(p, s), \operatorname{dist}(q, s))$. □

Now we are able to prove Lemma 3.3.

Proof of Lemma 3.3. Consider $x, y \in C_i$ such that $\operatorname{dist}(x, y) = D$. Consider arbitrary points $p \in C_i, s \in P \setminus C_i$. Since $\operatorname{dist}(p, x) + \operatorname{dist}(p, y) \geq \operatorname{dist}(x, y) = D$, we have either $\operatorname{dist}(p, x) \geq D/2$ or $\operatorname{dist}(p, y) \geq D/2$. Without loss of generality, we assume $\operatorname{dist}(p, x) \geq D/2$. According to Lemma B.1, we have $(\alpha/4 - 1/2)D \leq (\alpha/2 - 1)\operatorname{dist}(p, x) < \operatorname{dist}(p, s)$. □

B.2. Proof of Lemma 3.5

Proof of Lemma 3.5. Since we connect each vertex to at most one another vertex for each hash function h_i, the number of edges $|E| \leq n \cdot t = O(\mathbf{p}_1^{-1} n \log n)$. Consider two points $p, q \in P$ with $\operatorname{dist}(p, q) \leq D$. By Definition 3.4, $\forall i \in [t]$, the probability that $h_i(p) = h_i(q)$ is at least $\mathbf{p}_1$. Thus, the probability that $\nexists i \in [t], h_i(p) = h_i(q)$ is at most $(1 - \mathbf{p}_1)^t \leq 1/n^5$. If $\exists i \in [t], h_i(p) = h_(q)$, p, q are either connected or have a common neighbor. Next, consider two points $p, q \in P$ with $\operatorname{dist}(p, q) \geq cD$. By union bound, the probability that $\exists i \in [t], h_i(p) = h_i(q)$ is at most $1 - t\,\mathbf{p}_2$. Notice that there is an edge between p, q only when $\exists i \in [t], h_i(p) = h_i(q)$. By taking a union bound over all pairs $p, q \in P$, we conclude the proof. □

B.3. Proof of Theorem 3.6

Proof of Theorem 3.6. We suppose that all subroutines which call Algorithm 1 succeed. By union bound, this happens with probability at least $1 - O(\mathbf{p}_3 \log \Delta)$.

Consider an arbitrary cluster $C \in \mathcal{C}$. Let $D = \max_{p,q \in C} \operatorname{dist}(p, q)$. We can find $i^* \in \{0, 1, \cdots, L = \log_{1.01} \Delta\}$ such that $1.01D \geq 1.01^{i^*} \geq D$. According to Lemma 3.5, if there is an edge between p, q in G'_{i^*}, then $\operatorname{dist}(p, q) < \alpha' \cdot 1.01^{i^*} \leq (\alpha/4 - 1/2)D$. Due to Lemma 3.3, $\forall p \in C, q \in P \setminus C$, we have $\operatorname{dist}(p, q) > (\alpha/4 - 1/2)D$ which implies that there is no edge between p, q in G'_{i^*}. In addition, according to Lemma 3.5, $\forall p, q \in C$, since $\operatorname{dist}(p, q) \leq D \leq 1.01^{i^*}$, p, q are either connected in G'_{i^*} or have a common neighbor in G'_{i^*}. Thus, C is a connected component in G'_{i^*} and its diameter is at most 2.

According to Lemma 3.5, if there is an edge between p, q in G'_0, then $\operatorname{dist}(p, q) < \alpha'$. Therefore, each connected component of G'_0 is a singleton, and it implies that $\forall p \in P, \{p\} \in \mathcal{Q}_0$. According to Lemma 3.5 again, since $\forall p, q \in P, \operatorname{dist}(p, q) \leq \Delta$, we have $\forall p, q \in P$, either there is an edge between p, q in G'_L or p, q have a common neighbor in G'_L. Thus, G'_L is connected, and the diameter of G'_L is at most 2. Therefore, $P \in \mathcal{Q}_L$. □

B.4. Proof of Lemma 3.7

Proof of Lemma 3.7. With probability at least $1 - 2\,\mathbf{p}_3$, the subroutines in Algorithm 2 which call Algorithm 1 for $i = 0$ and $i = L$ succeed. According to Lemma 3.5, if there is an edge between p, q in G'_0, then $\operatorname{dist}(p, q) < \alpha'$. Therefore, each

connected component of G'_0 is a singleton, and it implies that $\forall p \in P, \{p\} \in \mathcal{Q}_0 \subseteq \mathcal{Q}$. According to Lemma 3.5 again, since $\forall p, q \in P, \text{dist}(p, q) \leq \Delta$, we have $\forall p, q \in P$, either there is an edge between p, q in G'_L or p, q have a common neighbor in G'_L. Thus, G'_L is connected, and the diameter of G'_L is at most 2. Therefore, $P \in \mathcal{Q}_L \subseteq \mathcal{Q}$.

Since $\forall i' > i \in \{0, 1, \cdots, L\}$ and for each connected component C in G'_i there is at most one connected component C' in $G'_{i'}$ satisfying $C \subseteq C'$, each $C \in \mathcal{Q}_i$ can have at most one parent in $\mathcal{T}$. Thus, $\mathcal{T}$ is a foreset. Furthermore, since $P \in \mathcal{Q}_L$, P should be the unique root of $\mathcal{T}$ which implies that $\mathcal{T}$ is a tree.

Consider an arbitrary $C \in \mathcal{Q}_i$. The parent of C must be in $\mathcal{Q}_{i'}$ for some $i' > i$. Thus, the depth of $\mathcal{T}$ is at most L.

□

B.5. Proof of Lemma 3.9

Proof of Lemma 3.9. Let us first prove that any pruning set $\mathcal{S}$ of $\mathcal{T}$ is a partition of P. According to the definition of pruning set and the construction of the parent of each vertex in $\mathcal{T}$, for each leaf $\{p\} \subset P$ we can find $S \in \mathcal{S}$ such that $\{p\} \subseteq S$, i.e., $p \in S$. Therefore, $\bigcup_{S \in \mathcal{S}} S = P$. Next we only need to show that $\forall S \neq S' \in \mathcal{S}$, $S \cap S' = \emptyset$. We prove it by contradiction. Suppose $S \cap S' \neq \emptyset$. We know that S is a connected component of G'_i for some i and S' is a connected component of $G'_{i'}$ for some i'. Since $S \neq S'$, we have $i \neq i'$. Without loss of generality we assume $i' > i$. Let S'' be S or an ancestor of S such that S'' is a connected component of $G'_{i''}$ for some $i'' < i'$ and the parent of S'' is a connected component in $G'_{i'''}$ for $i''' \geq i'$. By the construction of $\mathcal{T}$, we know that $S \subseteq S''$. Thus, $S'' \cap S' \neq \emptyset$. According to Algorithm 2, the edge set of $G'_{i''}$ is a subset of $G'_{i'}$. Thus, if two points are in the same connected component in $G'_{i''}$, they must be in the same connected component in $G'_{i'}$. Since S' is a connected component of $G'_{i'}$ and S'' is a connected component of $G'_{i''}$, we have $S'' \subseteq S'$. Thus, the parent of S'' must be S' which implies that S' is an ancestor of S. It contradicts to that $\mathcal{S}$ is a pruning set of $\mathcal{T}$.

Next we show that any partition $\mathcal{C} = \{C_1, C_2, \cdots, C_s\}$ of P satisfying $\forall C \in \mathcal{C}, C \in \mathcal{Q}$ is a pruning set of $\mathcal{T}$. Since $\forall C \neq C' \in \mathcal{C}$ are disjoint, C cannot be an ancestor of C' in $\mathcal{T}$. Thus, we only need to show that for any $p \in P$, there exists $C \in \mathcal{C}$ such that C is on the path from leaf $\{p\}$ to the root in $\mathcal{T}$. Consider a point $p \in P$. Since $\mathcal{C}$ is a partition of P, we can find $C \in \mathcal{C}$ such that $p \in C$. If $C = \{p\}$, we are done. Otherwise, we are going to show that C is an ancestor of $\{p\}$ in $\mathcal{T}$. Suppose C is a connected component of G'_i for some $i \in \{0, 1, \cdots, L\}$. Let S be $\{p\}$ or an ancestor of $\{p\}$ such that S is a connected component in $G'_{i'}$ for $i' < i$ and the parent of S is a connected component of $G'_{i''}$ for $i'' \geq i$. According to Algorithm 2, the edge set of $G'_{i'}$ is a subset of G'_i. Thus, the points that are in the same connected component as p in $G'_{i'}$ are also in the same connected component as p in G_i. Therefore, $S \subseteq C$. By the construction of the parent of S, C must be the parent of S. Therefore, C is an ancestor of $\{p\}$ in $\mathcal{T}$. □

B.6. Proof of Lemma 3.10

Proof of Lemma 3.10. First let us show that each vertex in $\mathcal{T}'$ can have at most 2 children. Each leaf vertex in $\mathcal{T}$ is still a leaf vertex in $\mathcal{T}'$. Consider a non-leaf vertex C and its children $C_1, C_2, \cdots, C_s$ in $\mathcal{T}$. In Algorithm 4, we add s auxiliary vertices $C'_1, \cdots, C'_s$. C has exactly one child C'_1. We conceptually regard $C_1, C_2, \cdots, C_s$ as $C'_{s+1}, C'_{s+2}, \cdots, C'_{2s}$. Then each auxiliary vertex $C'_i, i \in [s]$ can have at most 2 children: $C'_{i\cdot 2}$ and $C'_{i\cdot 2+1}$. It does not add any additional children to $C_1, C_2, \cdots, C_s$. Therefore, each vertex in the tree $\mathcal{T}'$ outputted by Algorithm 4 has at most two children.

Consider two vertices C and C' in $\mathcal{T}$ such that C is the parent of C'. In Algorithm 4, we will add at most $O(\log n)$ vertices in the path between C' and C in tree $\mathcal{T}'$. Thus, the depth of

Then, let us show that any pruning set of $\mathcal{T}$ is also a pruning set of $\mathcal{T}'$. According to Algortihm 4, the set of leaf vertices of $\mathcal{T}'$ is exactly the same as the set of leaf vertices of $\mathcal{T}$. Consider a path from a leaf to the root in $\mathcal{T}'$. The difference from the path from the leaf to the root in $\mathcal{T}$ is that there are may be some auxiliary vertices inserted into the path. Since a pruning set $\mathcal{S}$ of $\mathcal{T}$ does not contain any auxiliary vertex, for each leaf of $\mathcal{T}'$, there is still a unique vertex in $\mathcal{S}$ that is on the path from the leaf to the root in $\mathcal{T}'$.

Now suppose that any pruning set of $\mathcal{T}$ is a partition of P. We will prove that any pruning set of $\mathcal{T}'$ is a partition of P. Consider an arbitary pruning set $\mathcal{S}'$ of $\mathcal{T}'$. We construct a pruning set $\mathcal{S}$ of $\mathcal{T}$ in the following way. If $S' \in \mathcal{S}'$ is also a vertex in $\mathcal{T}$, we add it to $\mathcal{S}$ directly. If $S' \in \mathcal{S}'$ is an auxiliary vertex, according to the construction of S' in Algorithm 4, we can find a set of vertices $S_1, S_2, \cdots, S_t$ in $\mathcal{T}$ such that (1). $S' = S_1 \cup S_2 \cup \cdots \cup S_t$, (2). for any leaf that is in the subtree of S' in $\mathcal{T}'$, there is a unique $i \in [t]$ that the leaf is in the subtree of S_i in $\mathcal{T}'$. We add $S_1, S_2, \cdots, S_t$ into $\mathcal{S}$. Consider each leaf vertex in $\mathcal{T}$. It is also a leaf vertex in $\mathcal{T}'$. Since $\mathcal{S}'$ is a pruning set of $\mathcal{T}'$, we can find a unique $S' \in \mathcal{S}'$ which is on the

path from the leaf to the root in $\mathcal{T}'$. By our construction of $\mathcal{S}$, we can find a unique $S \in \mathcal{S}$ which is on the path from the leaf to the root in $\mathcal{T}$. Thus, $\mathcal{S}$ is a pruning set of $\mathcal{T}$ which implies that $\mathcal{S}$ is a partition of P. Since each cluster of $\mathcal{S}'$ is either a cluster in $\mathcal{S}$ or a union of several clusters in $\mathcal{S}$, $\mathcal{S}'$ is also a partition of P. □

B.7. Proof of Lemma 3.11

To prove Lemma 3.11, we first show the following two useful lemmas.

Lemma B.2 (Cost of optimal pruning set)**.** *For vertex C in $\mathcal{T}'$ and $i \in [k]$, $\mathrm{DPcost}(C, i)$ computed by Algorithm 5 satisfies $\mathrm{DPcost}(C, i) = \min_{\text{pruning set } \mathcal{S} \text{ of the subtree rooted at } C: |\mathcal{S}| \le i} \sum_{C' \in \mathcal{S}} w(C')$.*

Proof. The proof is by an induction over the depth of the subtree rooted at C. Consider the base case when the depth of the subtree rooted at C is 0, i.e., C is a leaf vertex. In this case, the subtree of C is a singleton and has only one pruning set which is $\{C\}$. Since $\forall i \in [k], \mathrm{DPcost}(C, i) = w(C)$, the lemma statement holds for C.

Suppose the lemma statement is true for all vertices whose subtree depth is at most $H - 1$. Now consider a vertex C such that the subtree rooted at C has depth H. Consider $i \in [k]$. If C only has one child C', then a pruning set with size at most i of the subtree rooted at C is either $\{C\}$ or a pruning set with size at most i of the subtree rooted at C'. Thus the lemma statement holds since $\mathrm{DPcost}(C, i) = \min(w(C), \mathrm{DPcost}(C', i))$. If C has two children C' and C'', then a pruning set with size at most i of the subtree rooted at C is either $\{C\}$ or the union of $\mathcal{S}'$ and $\mathcal{S}''$ where $|\mathcal{S}'| + |\mathcal{S}''| \le i$, and $\mathcal{S}'$ and $\mathcal{S}''$ are a pruning set of the subtree rooted at C' and a pruning set of the subtree rooted at C'' respectively. By optimality, the lemma statement holds since $\mathrm{DPcost}(C, i) = \min(w(C), \min_{i' \in [i-1]} \mathrm{DPcost}(C', i') + \mathrm{DPcost}(C'', i - i'))$. □

Lemma B.3 (Traverse backwards via dynamic programming values)**.** *At the beginning (or the end) of each repetition of line 5 of Algorithm 5, the following invariants hold:*

1. $|\mathcal{S}| + \sum_{(C,i) \in \mathcal{L}} i \le k$.
2. $\sum_{C \in \mathcal{S}} w(C) + \sum_{(C,i) \in \mathcal{L}} \mathrm{DPcost}(C, i) = \mathrm{DPcost}(R, k)$, *where R is the root of $\mathcal{T}'$.*

Proof. Let us first prove the first invariant. Before the first repetition, $\mathcal{S} = \emptyset$ and $\mathcal{L} = \{(R, k)\}$. Thus, the invariant holds. Suppose the invariant holds before an repetition of line 5 of Algorithm 5. Now consider the changes during the repetition. We first remove (C, i) from $\mathcal{L}$, thus the target value decreased by i. If we add C into $\mathcal{S}$, then the target value increased by 1. The total change is at most $1 - i \le 0$. If we add (C', i) into $\mathcal{L}$, the total change is at most $i - i = 0$. If we add (C', i') and $(C'', i - i')$ into $\mathcal{L}$, the total change is at most $i' + i - i' - i = 0$. Therefore, in any case, the first invariant always holds.

Let us consider the second invariant. Before the first repetition, $\mathcal{S} = \emptyset$ and $\mathcal{L} = \{(R, k)\}$. Thus, the invariant holds. Suppose the invariant holds before an repetition. Now consider the changes during the repetition. We first remove (C, i) from $\mathcal{L}$, thus the target value decreased by $\mathrm{DPcost}(C, i)$. If $\mathrm{DPcost}(C, i) = w(C)$, we add C into $\mathcal{S}$, thus the target value increased by $w(C) = \mathrm{DPcost}(C, i)$. The total change is $w(C) - \mathrm{DPcost}(C, i) = 0$. If C has one child C' and we add (C', i) into $\mathcal{L}$, we know that $\mathrm{DPcost}(C, i) = \mathrm{DPcost}(C', i)$. The total change is $\mathrm{DPcost}(C', i) - \mathrm{DPcost}(C, i) = 0$. If C has two children C', C'' and we add $(C', i'), (C'', i - i')$ into $\mathcal{L}$, we know that $\mathrm{DPcost}(C, i) = \mathrm{DPcost}(C', i') + \mathrm{DPcost}(C'', i - i')$. The total change is $\mathrm{DPcost}(C', i') + \mathrm{DPcost}(C'', i - i') - \mathrm{DPcost}(C, i) = 0$. Therefore, in any case, the second invariant always holds. □

Proof of Lemma 3.11. Let R be the root of $\mathcal{T}'$. According to Lemma B.2, $\mathrm{DPcost}(R, k) = \min_{\text{pruning set } \mathcal{S}' \text{ of } \mathcal{T}': |\mathcal{S}'| \le k} \sum_{C \in \mathcal{S}'} w(C)$. According to Lemma B.3, the output $\mathcal{S}$ has size at most k and $\sum_{C \in \mathcal{S}} w(C) = \mathrm{DPcost}(R, k) = \min_{\text{pruning set } \mathcal{S}' \text{ of } \mathcal{T}': |\mathcal{S}'| \le k} \sum_{C \in \mathcal{S}'} w(C)$. □

B.8. Proof of Lemma 3.12

To prove Lemma 3.12, we need to prove the following two useful lemmas:

Lemma B.4 (Size of near optimal pruning set)**.** *For vertex C in $\mathcal{T}'$ and $i \in \{-1, 0, \cdots, l\}$, $\mathrm{DPsize}(C, i)$ computed by Algorithm 6 satisfies*

1. *There is a pruning set $\mathcal{S}$ with $|\mathcal{S}| = \mathrm{DPsize}(C, i)$ of the subtree rooted at C such that $\sum_{C' \in \mathcal{S}} w(C') \le \lambda_i$.*
2. *Every pruning set $\mathcal{S}$ with $|\mathcal{S}| < \mathrm{DPsize}(C, i)$ of the subtree rooted at C must have $\sum_{C' \in \mathcal{S}} w(C') > (1+\varepsilon')\lambda_i/(1+\varepsilon)$.*

Proof. The proof is by an induction over the depth of the subtree rooted at C. Consider the base case when the depth of the subtree rooted at C is 0., i.e., C is a leaf vertex. In this case, the subtree of C is a singleton and has only one pruning set which is $\{C\}$. The weight $w(C) = 0$. Since $\forall i \in \{-1, 0, \cdots, l\}$, $\mathrm{DPsize}(C, i) = 1$, the pruning set with size less than $\mathrm{DPsize}(C, i) = 1$ does not exist, and there is a pruning set $\{C\}$ of the subtree rooted at C such that $w(C) = 0 \leq \lambda_i$.

Suppose for any vertex $\hat{C}$ of which subtree depth is at most $M-1$ and any $i \in \{-1, 0, \cdots, l\}$, there is a pruning set $\hat{\mathcal{S}}$ with $\hat{\mathcal{S}} = \mathrm{DPsize}(\hat{C}, i)$ of the subtree rooted at $\hat{C}$ such that $\sum_{C' \in \hat{\mathcal{S}}} w(C') \leq \lambda_i$. Now we consider a vertex C of which subtree depth is M. If $\lambda_i \geq w(C)$, we have $\mathrm{DPsize}(C, i) = 1$, and $\{C\}$ is a pruning set of the subtree rooted at C and $w(C) \leq \lambda_i$. Consider the case that C only has one child C'. We have $\mathrm{DPsize}(C, i) = \mathrm{DPsize}(C', i)$ for $\lambda_i < w(C)$. By induction hypotheis, since the depth of the subtree rooted at C' is at most $M-1$, we can find a prunning set $\mathcal{S}$ with $|\mathcal{S}| = \mathrm{DPsize}(C', i)$ of the subtree rooted at C' such that $\sum_{\tilde{C} \in \mathcal{S}} w(\tilde{C}) \leq \lambda_i$. Since a pruning set of the subtree rooted at C' is also a pruning set of the subtree rooted at C, $\mathcal{S}$ is a pruning set with $|\mathcal{S}| = \mathrm{DPsize}(C, i)$ of the subtree rooted at C such that $\sum_{\tilde{C} \in \mathcal{S}} w(\tilde{C}) \leq \lambda_i$. Consider the case that C has two children C' and C''. We have $\mathrm{DPsize}(C, i) = \mathrm{DPsize}(C', i') + \mathrm{DPsize}(C'', i'')$ for $\lambda_i < w(C)$ and some i', i'' satisfying $\lambda_{i'} + \lambda_{i''} \leq \lambda_i$. By induction hypothesis, since the depth of the subtree rooted at C' is at most $M-1$ and the depth of the subtree rooted at C'' is also at most $M-1$, we can find a pruning set $\mathcal{S}'$ with $|\mathcal{S}'| = \mathrm{DPsize}(C', i')$ of the subtree rooted at C' such that $\sum_{\tilde{C} \in \mathcal{S}'} w(\tilde{C}) \leq \lambda_{i'}$ and a pruning set $\mathcal{S}''$ with $|\mathcal{S}''| = \mathrm{DPsize}(C'', i'')$ of the subtree rooted at C'' such that $\sum_{\tilde{C} \in \mathcal{S}''} w(\tilde{C}) \leq \lambda_{i''}$. Thus, $\mathcal{S}' \cup \mathcal{S}''$ is a pruning set with $|\mathcal{S}' \cup \mathcal{S}''| = \mathrm{DPsize}(C', i') + \mathrm{DPsize}(C'', i'') = \mathrm{DPsize}(C, i)$ of the subtree rooted at C such that $\sum_{\tilde{C} \in \mathcal{S}' \cup \mathcal{S}''} w(\tilde{C}) \leq \lambda_{i'} + \lambda_{i''} \leq \lambda_i$.

Next suppose for any vertex $\hat{C}$ of which subtree depth is at most $M-1$ and any $i \in \{-1, 0, \cdots, l\}$, every pruning set $\hat{\mathcal{S}}$ with $|\hat{\mathcal{S}}| < \mathrm{DPsize}(C, i)$ of the subtree rooted at $\hat{C}$ must have $\sum_{C' \in \hat{\mathcal{S}}} w(C') > \lambda_i/(1+\varepsilon')^{M-1}$. Recall Algorithm 6 that $\varepsilon' = \varepsilon/(2H)$ where H is the depth of $\mathcal{T}'$. Now we consider a vertex C of which subtree depth is M and consider $i \in \{-1, 0, \cdots, l\}$. Suppose there is a pruning set $\mathcal{S}$ with $|\mathcal{S}| < \mathrm{DPsize}(C, i)$ of the subtree rooted at C such that $\sum_{\tilde{C} \in \mathcal{S}} w(\tilde{C}) \leq \lambda_i/(1+\varepsilon')^M$. If $\mathcal{S} = \{C\}$, we have $w(C) \leq \lambda_i$. According to Algorithm 6, we have $\mathrm{DPsize}(C, i) = 1$ which contradicts to $|\mathcal{S}| < \mathrm{DPsize}(C, i)$. Consider the case that C has one child C'. The depth of the subtree rooted at C' is at most $M-1$. Since $\mathcal{S} \neq \{C\}$, $\mathcal{S}$ is also a pruning set with $|\mathcal{S}| < \mathrm{DPsize}(C, i) = \mathrm{DPsize}(C', i)$ of the subtree rooted at C' such that $\sum_{\tilde{C} \in \mathcal{S}} w(\tilde{C}) \leq \lambda_i/(1+\varepsilon')^M \leq \lambda_i/(1+\varepsilon')^{M-1}$ which contradicts to the induction hypothesis. Consider the case that C has children C' and C''. The depth of the subtree rooted at C' is at most $M-1$ and the depth of the subtree rooted at C'' is also at most $M-1$. Since $\mathcal{S} \neq \{C\}$, $\mathcal{S}$ can be written as $\mathcal{S}' \cup \mathcal{S}''$ where $\mathcal{S}'$ is a pruning set of the subtree rooted at C' and $\mathcal{S}''$ is a pruning set of the subtree rooted at C''. We can find $i', i'' \in \{-1, 0, \cdots, l\}$ such that $(1+\varepsilon')^M \sum_{\tilde{C} \in \mathcal{S}'} w(\tilde{C}) \geq \lambda_{i'} \geq (1+\varepsilon')^{M-1} \sum_{\tilde{C} \in \mathcal{S}'} w(\tilde{C})$ and $(1+\varepsilon')^M \sum_{\tilde{C} \in \mathcal{S}''} w(\tilde{C}) \geq \lambda_{i''} \geq (1+\varepsilon')^{M-1} \sum_{\tilde{C} \in \mathcal{S}''} w(\tilde{C})$. By induction hypothesis, we have $\mathrm{DPsize}(C', i') \leq |\mathcal{S}'|$ and $\mathrm{DPsize}(C'', i'') \leq |\mathcal{S}''|$. Since $\lambda_{i'} + \lambda_{i''} \leq (1+\varepsilon')^M \sum_{\tilde{C} \in \mathcal{S}} w(\tilde{C}) \leq \lambda_i$, we have $\mathrm{DPsize}(C, i) \leq \mathrm{DPsize}(C', i') + \mathrm{DPsize}(C'', i'') \leq |\mathcal{S}'| + |\mathcal{S}''| = |\mathcal{S}|$ according to Algorithm 6, which contradicts to $|\mathcal{S}| < \mathrm{DPsize}(C, i)$.

Thus, for every vertex C in $\mathcal{T}'$ and $i \in \{-1, 0, \cdots, l\}$, every pruning set $\mathcal{S}$ with $|\mathcal{S}| < \mathrm{DPsize}(C, i)$ of the subtree rooted at C must have $\sum_{C' \in \mathcal{S}} w(C') > \lambda_i/(1+\varepsilon')^H \geq (1+\varepsilon')\lambda_i/(1+\varepsilon)$. □

Lemma B.5 (Traverse backwards via approximate dynamic programming values)**.** *At the beginning (or the end) of each repetition of line 6 of Algorithm 6, the following invariants hold:*

1. $|\mathcal{S}| + \sum_{(C,i) \in \mathcal{L}} \mathrm{DPsize}(C, i) \leq k$.
2. $\sum_{C \in \mathcal{S}} w(C) + \sum_{(C,i) \in \mathcal{L}} \lambda_i \leq \lambda_{i^*}$.

Proof. Let us first prove the first invariant. Before the first repetition, $\mathcal{S} = \emptyset$ and $\mathcal{L} = \{(R, i^*)\}$. Since we choose i^* such that $\mathrm{DPsize}(R, i^*) \leq k$, the invariant holds. Suppose the invariant holds before an repetition. Now consider the changes during the repetition. We first remove (C, i) from $\mathcal{L}$, thus the target value decreased by $\mathrm{DPsize}(C, i)$. If we add C into $\mathcal{S}$, then $\mathrm{DPsize}(C, i) = 1$ and $|\mathcal{S}|$ increased by 1. The total change is $1 - \mathrm{DPsize}(C, i) = 0$. If we add (C', i) into $\mathcal{L}$, the total change is at most $\mathrm{DPsize}(C', i) - \mathrm{DPsize}(C, i) = 0$.. If we add (C', i') and (C'', i'') into $\mathcal{L}$, the total change is at most $\mathrm{DPsize}(C', i') + \mathrm{DPsize}(C'', i'') - \mathrm{DPsize}(C, i) = 0$. Therefore, in any case, the first invariant always holds.

Let us consider the second invariant. Before the first repetition, $\mathcal{S} = \emptyset$ and $\mathcal{L} = \{(R, i^*)\}$. Thus, the invariant holds. Suppose the invariant holds before an repetition. Now consider the changes during the repetition. We first remove (C, i) form $\mathcal{L}$, thus the target value decreased by λ_i. If $\mathrm{DPsize}(C, i) = 1$, we have $w(C) \leq \lambda_i$, and we add C into $\mathcal{S}$. Thus, the target value cannot increase when $\mathrm{DPsize}(C, i) = 1$. If C has only one child C', then we add (C', i) into $\mathcal{L}$. The

total change is $\lambda_i - \lambda_i = 0$. If C has children C' and C'', then we add (C', i') and (C'', i'') into $\mathcal{L}$. The total change is $\lambda_{i'} + \lambda_{i''} - \lambda_i \leq 0$. Thus, the invariant always holds. □

Now we are able to prove Lemma 3.12.

Proof of Lemma 3.12. According to Lemma B.5, the output $\mathcal{S}$ of Algorithm 6 is a pruning set of $\mathcal{T}'$ such that $|\mathcal{S}| \leq k$ and $\sum_{C\in\mathcal{S}} w(C) \leq \lambda_{i^*}$. If $i^* = -1$, we have $\sum_{C\in\mathcal{S}} w(C) = 0$.

By our choice of i^*, we have $\mathrm{DPsize}(R, i^* - 1) > k$. According to Lemma B.4, we have $\min_{\text{pruning set } \mathcal{S}' \text{ of } \mathcal{T}':|\mathcal{S}'|\leq k} \sum_{C\in\mathcal{S}'} w(C) > (1+\varepsilon')\lambda_{i^*-1}/(1+\varepsilon)$. If $i^* = 0$, we know that $\min_{\text{pruning set } \mathcal{S}' \text{ of } \mathcal{T}':|\mathcal{S}'|\leq k} \sum_{C\in\mathcal{S}'} w(C) > 0$. Since the minimum positive weight is at least 1, we have $\sum_{C\in\mathcal{S}} w(C) = 1 = \min_{\text{pruning set } \mathcal{S}' \text{ of } \mathcal{T}':|\mathcal{S}'|\leq k} \sum_{C\in\mathcal{S}'} w(C)$. If $i^* > 0$, we have $\sum_{C\in\mathcal{S}} w(C) \leq \lambda_{i^*} \leq (1+\varepsilon')\lambda_{i^*-1} \leq (1+\varepsilon) \cdot \min_{\text{pruning set } \mathcal{S}' \text{ of } \mathcal{T}':|\mathcal{S}'|\leq k} \sum_{C\in\mathcal{S}'} w(C)$. □

B.9. Guarantees of Offline Algorithms

Now we are able to conclude the guarantees of our k-means algorithms in the offline setting. Let $\mathcal{C}^*$ be the optimal k-means clustering for P. The success probability of Algorithm 2 is given in Theorem 3.6. We condition on that Algorithm 2 succeeds. According to Theorem 3.6 and Lemma 3.9, $\mathcal{C}^*$ is a pruning set of $\mathcal{T}$ and any pruning set of $\mathcal{T}$ is a partition of P. According to Lemma 3.10, $\mathcal{C}^*$ is also a pruning set of $\mathcal{T}'$ and any pruning set of $\mathcal{T}'$ is also a partition of P. Because $w(C) = \sum_{p\in C} \mathrm{dist}^2(p, \mathrm{mean}(C))$, for any $\beta \geq 1$, if $\mathcal{C}$ is a pruning set of $\mathcal{T}'$ with $|\mathcal{C}| \leq k$ and $\sum_{C\in\mathcal{C}} w(C) \leq \beta \cdot \sum_{C\in\mathcal{C}^*} w(C)$, then we know that $\mathcal{C}$ is β-approximate k-means clustering. According to Lemma 3.11, Algorithm 5 outputs the optimal k-means clustering for P. According to Lemma 3.12, Algorithm 3.12 outputs $(1+\varepsilon)$-approximate k-means clustering for P. Now let us consider the running time of algorithms. Suppose the running time to compute all LSH value $h_i(p)$ for $p \in P$ and $i \in [t]$ in Algorithm 1 is at most T. The running time to build G in Algorithm 1 is at most $O(T \log n)$ because we can use sorting to group the points with the same hash value together and the number of edges should be at most T. Recall $L = O(\log \Delta)$ and Δ is an upper bound of the aspect ratio of P. The running time to build all $G_0, G_1, \cdots, G_L$ in Algorithm 2 is at most $O(TL \log n)$. Since each G_i has size at most T, each G'_j has size at most TL. The running time to find connected components for all $G'_0, G'_1, \cdots, G'_L$ is at most $O(TL^2)$. Thus, the running time of Algorithm 2 is at most $O(TL^2 + TL \log n)$. Since each connected component of G'_i is a subset of a connected component of $G'_{i'}$ for $i' > i$, we only need $O(L)$ time to find the parent of C for each $C \in \mathcal{Q}_0, \mathcal{Q}_1, \cdots, \mathcal{Q}_L$ in Algorithm 3. Since $\forall i \in \{0, 1, \cdots, L\}, |\mathcal{Q}_i| \leq n$, the running time of Algorithm 3 is at most $O(nL^2)$. The number of vertices in tree $\mathcal{T}$ is at most $O(nL)$ and depth of $\mathcal{T}$ is $O(L)$. The running time to create auxiliary vertices for each non-leaf vertex of $\mathcal{T}$ in Algorithm 4 is linear in the number of its children. Thus, the total running time of Algorithm 4 is $O(nL)$. The number of vertices in $\mathcal{T}'$ is $O(nL)$ and depth of $\mathcal{T}'$ is $O(L \log n)$. To compute a single $\mathrm{DPcost}(C, i)$ in Algorithm 5, we need $O(k)$ time. Thus, the total running time of Algorithm 5 is at most $O(nLk^2)$. Therefore, to compute the optimal k-means clustering, the overall running time is at most $(T + nk^2)\mathrm{poly}(L \log n)$. In Algorithm 6, we have $l = O(\varepsilon^{-1}(L + \log n)L \log n)$. Since $\mathrm{DPsize}(C, i)$ is monotonic over i, to compute a single $\mathrm{DPsize}(C, i)$ in Algorithm 6, we only need $O(l)$ time. Thus, the total running time of Algorithm 6 is at most $O(nLl^2)$. Therefore, the overall running time to compute $(1+\varepsilon)$-approximate k-means clustering is at most $(T + n/\varepsilon^2)\mathrm{poly}(L \log n)$. If we use the LSH of (Andoni & Indyk, 2006), then $T \leq n^{1+O(1/\alpha^2)+o(1)}d$. If we use the LSH of (Datar et al., 2004), then $T \leq n^{1+O(1/\alpha)+o(1)}d$.

C. Missing Details of Section 4

C.1. Proof of Lemma 4.2

Proof of Lemma 4.2. Let $s = 5c^2/(c^2-1)$. We have $s \in [1, \log n]$. Let $\mathcal{H}$ be the $(D, cD, 1/n^{s/c^2+o(1)}, 1/n^s)$-sensitive hash family described in Lemma 4.1. We can handle $h_1, h_2, \cdots, h_t$ described in Algorithm 1 simultaneously. According to Algorithm 1, we have $t = n^{s/c^2+o(1)} = n^{5/(c^2-1)+o(1)}$. According to Lemma 4.1, we can compute $h_i(p)$ for every $p \in P$ and $i \in [t]$ in $O(1)$ rounds and $n^{1+5/(c^2-1)+o(1)}d$ total space. Furthermore, the algorithm is fully scalable.

Then we simultaneously handle each $i \in [t]$. Consider $i \in [t]$, we can sort all points $p \in P$ by their hash value $h_i(p)$. According to Theorem 2.2, sorting in the MPC model takes $O(1)$ rounds and the total space needed for one i is at most $O(n \log^3 n)$ (each hash value takes $O(\log^3 n)$ space according to Lemma 4.1). Thus, we can find the point with the smallest index for each hash values $h_i(p)$. Therefore, we can create an edge between p and q where q is the point with the smallest

index satisfying $h_i(q) = h_i(p)$. Thus, the overall algorithm is fully scalable and takes $O(1)$ rounds. The total space needed is at most $n^{1+5/(c^2-1)+o(1)}d$.

Since $\mathcal{H}$ is a $(D, cD, 1/n^{s/c^2+o(1)}, 1/n^s)$-sensitive hash family and $s = 5c^2/(c^2-1)$, the success probability of Algorithm 1 is at least $1 - O(1/n^2)$ according to Lemma 3.5. Since each edge in the output G corresponds to a point $p \in P$ and an $i \in [t]$, the number of edges of G is at most $O(nt) = n^{1+5/(c^2-1)+o(1)}$. □

C.2. Proof of Lemma 4.3

Proof of Lemma 4.3. We run Algorithm 1 for all $i \in \{0, 1, \cdots, L\}$ simultaneously. According to Lemma 4.2, the number of rounds is $O(1)$, the total space needed is $n^{1+5/(\alpha'^2-1)+o(1)}Ld = n^{1+5/(\alpha'^2-1)+o(1)}d$, and the overall success probability is at least $1 - O(L/n^2) = 1 - O(1/n^{1.99})$. In addition, the procedure is fully scalable. Furthermore, according to Lemma 4.2, the size of each $G_0, G_1, \cdots, G_L$ is at most $n^{1+5/(\alpha'^2-1)+o(1)}$.

For each edge e in G_i, we can make $i+1$ copies of the edge. Then we add each copy into each graph $G'_0, G'_1, \cdots, G'_i$. Since $i \le L = n^{o(1)}$, this can be done locally on machines. The size of each $G'_0, G'_1, \cdots, G'_i$ is at most $n^{1+5/(\alpha'^2-1)+o(1)}L = n^{1+5/(\alpha'^2-1)+o(1)}$.

Next we compute $\mathcal{Q}_i$ for all $i \in \{0, 1, \cdots, L\}$ simultaneously. Consider $i \in \{0, 1, \cdots, L\}$. In the following, we show how to compute $\mathcal{Q}_i$ satisfying the properties listed in Algorithm 2. We use the following procedure to find connected components with diameter at most 2 in G'_i:

1. Initialize $\mathrm{label}(u) = u$ for each vertex u in G'_i.
2. Run 2 iterations: in each iteration, for each edge $\{u, v\}$ in G'_i, if $\mathrm{label}(u) < \mathrm{label}(v)$, update $\mathrm{label}(v) \leftarrow \mathrm{label}(u)$, or if $\mathrm{label}(v) < \mathrm{label}(u)$, update $\mathrm{label}(u) \leftarrow \mathrm{label}(v)$. If $\mathrm{label}(x)$ is updated by multiple $\mathrm{label}(y)$ in an iteration, take the minimum one.
3. For each vertex u in G'_i, let $C_u = \{v \mid \mathrm{label}(v) = u\}$. If $u = \mathrm{label}(u)$ and there is no edge $\{x, y\}$ in G'_i such that $\mathrm{label}(x) = u, \mathrm{label}(y) \neq u$, add C_u into $\mathcal{Q}_i$.

In the above procedure, we run two iterations of label propagation. Each iteration of label propagation can be done in $O(1)$ parallel time: for each edge $\{u, v\}$ in G'_i, we first query $\mathrm{label}(u)$ and $\mathrm{label}(v)$. This operation can be done in the CRCW PRAM model in $O(1)$ parallel time and thus can be simulated in MPC in $O(1)$ parallel time and the simulation is fully scalable (Goodrich et al., 2011). Next, we can use $\mathrm{label}(u)$ (or $\mathrm{label}(v)$) to update $\mathrm{label}(v)$ (or $\mathrm{label}(u)$) locally. Finally we can use sorting (Theorem 2.2) to keep the minimum updated value for each $\mathrm{label}(u)$. The total space needed for above label propagation step is at most the size of G'_i. Thus, it is at most $n^{1+5/(\alpha'^2-1)+o(1)}$.

Next, let us consider how to compute C_u and how to determine whether C_u should be added into $\mathcal{Q}_i$. C_u for all vertices u are easy to compute in MPC: for each vertex v, we locally add v into $C_{\mathrm{label}(v)}$. Thus, all C_u are constructed and stored distributedly. For each edge $\{x, y\}$ in G'_i, if $\mathrm{label}(x) \neq \mathrm{label}(y)$, we mark both $\mathrm{bad}(\mathrm{label}(x)) = \mathrm{bad}(\mathrm{label}(y)) = \mathrm{true}$. For each vertex u, we can query $\mathrm{label}(u)$ and $\mathrm{bad}(u)$. This operation can be done in CRCW PRAM in $O(1)$ parallel time and thus can be simulated in MPC in $O(1)$ parallel time and the simulation is fully scalable (Goodrich et al., 2011). If $\mathrm{label}(u) = u$ and $\mathrm{bad}(u) \neq \mathrm{true}$, then it implies that no edge $\{x, y\}$ in G'_i such that $\mathrm{label}(x) = u, \mathrm{label}(y) \neq u$. Then we just add C_u into $\mathcal{Q}_i$.

Therefore, above procedure has $O(1)$ overall parallel time and total space $n^{1+5/(\alpha'^2-1)+o(1)}$. In the remaining of the proof, we show that each $C \in \mathcal{Q}_i$ is a connected component of G'_i, and any connected component of G'_i with diameter at most 2 must be in $\mathcal{Q}_i$.

Let us first show that any $C_u \in \mathcal{Q}_i$ is a connected component. Notice that the label propagation process can only update $\mathrm{label}(\cdot)$ between connected vertices. Thus each $C_u \in \mathcal{Q}_i$ must be a subset of connected component. If $C_u \in \mathcal{Q}_i$ is not a connected component, we can find an edge $\{x, y\}$ such that $x \in C_u$ and $y \notin C_u$. Thus, We have $\mathrm{label}(x) = u$ and $\mathrm{label}(y) \neq u$ which contradicts to $C_u \in \mathcal{Q}_i$.

Next, let us show that if a connected component C in G'_i has diameter at most 2, C must be in $\mathcal{Q}_i$. Notice that in each iteration of label propagation, $\mathrm{label}(u)$ will be updated by the smallest label in its neighborhood. By induction, we can show that $\mathrm{label}(u) = \min_{v:\text{distance between} u,v \text{ in } G'_i \text{ is at most } 2} v$ for every vertex u in G'_i. Let u be the vertex in the connected component C with the smallest index. Since the diameter of C is at most 2, every vertex v in the connected component C has $\mathrm{label}(v) = u$. Thus, $C_u = C$ and $\mathrm{label}(u) = u$. Furthermore, since C is a connected component, there is no edge

$\{x, y\}$ such that $x \in C$ and $y \notin C$ which implies that there is no edge $\{x, y\}$ such that $\text{label}(x) = u, \text{label}(y) \neq u$. By our construction of $\mathcal{Q}_i$, we have $C \in \mathcal{Q}_i$. □

C.3. Proof of Lemma 4.4

Proof of Lemma 4.4. According to our representation of sets in the MPC model. For each $i \in \{0, 1, \cdots, L\}$, $C \in \mathcal{Q}_i$ and each $p \in C$, there is a pair $(\text{“}C\text{”}, p)$ stored in the system. Since $L = n^{o(1)}$, there are at most $n^{o(1)}$ pairs $(\text{“}C\text{”}, p)$ for p. Therefore, we can use sorting (Theorem 2.2) to group all $(\text{“}C_1\text{”}, p), (\text{“}C_2\text{”}, p), \cdots, (\text{“}C_s\text{”}, p)$ into a single machine, i.e., $\forall i \in \{0, 1, \cdots, L\}$ if $\exists C \in \mathcal{Q}_i$, $(\text{“}C\text{”}, p)$ is held by the machine. Suppose $\forall j \in [s], C_j \in \mathcal{Q}_{i_j}$ where $i_1 < i_2 < \cdots < i_s$.

Claim C.1. $C_1 = \{p\}$, $C_s = P$ and $\forall i \in [s-1], C_{i+1}$ is the parent of C_i in $\mathcal{T}$, i.e., $C_1, C_2, \cdots, C_s$ is a path from leaf $\{p\}$ to the root P in $\mathcal{T}$.

Proof. According to Lemma 3.7, condition on that Algorithm 2 succeeds, $\forall p \in P, p \in \mathcal{Q}_0$, and $P \in \mathcal{Q}_L$. Thus, we have $C_1 = \{p\}$ and $C_s = P$. According to Algorithm 2, a connected component of G'_i for $i \in \{0, 1, \cdots, L\}$ must be a subset of a connected component of $G'_{i'}$ for any $i' \geq i$. Since $p \in C_1, C_2, \cdots, C_s$, $\forall i \in [s-1]$, we have that C_{i+1} is the parent of C_i. Therefore, $C_1, C_2, \cdots, C_s$ is a path from leaf $\{p\}$ to the root P in $\mathcal{T}$. □

For the leaf-to-root path $C_1, C_2, \cdots, C_s$, we set the parent of C_i as C_{i+1} for each $i \in [s-1]$. We can use sorting (Theorem 2.2) to remove duplicated parent pointers. Since each vertex must be on at least one leaf-to-root path in $\mathcal{T}$, the parent of each vertex in $\mathcal{T}$ is computed. Since we only need to use sorting (Theorem 2.2), the overall parallel time is $O(1)$ and the total space is linear in $\sum_{i=0}^{L} |\mathcal{Q}_i| = O(nL) = n^{1+o(1)}$. Furthermore, the algorithm is fully scalable. □

C.4. Proof of Lemma 4.5

Proof of Lemma 4.5. Since we know the parent of each vertex in $\mathcal{T}$, we can sort (Theorem 2.2) all vertices in $\mathcal{T}$ by their parents, and rank each vertex among all children of its parent. We can also compute the number of children of each vertex. The parallel time is $O(1)$, the total space needed is $O(nL) = n^{1+o(1)}$, and the algorithm is fully scalable (see e.g., (Andoni et al., 2018)). Let $\text{rank}(C)$ be the rank of C among all children of the parent of C in $\mathcal{T}$. Let $\text{numchild}(C)$ be the number of children of C in $\mathcal{T}$.

According to Algorithm 4, each leaf of $\mathcal{T}$ is still a leaf of $\mathcal{T}'$. Thus, each leaf-to-root path in $\mathcal{T}$ corresponds to a leaf-to-root path in $\mathcal{T}'$. In addition, if C is the parent of C' in $\mathcal{T}$, C is still an ancestor of C' in $\mathcal{T}'$, and all vertices on the path between C' and C in $\mathcal{T}'$ are auxiliary vertices created when processing C in Algorithm 4. Suppose C' is the i-th children of C, i.e., $\text{rank}(C') = i$. Suppose $C'_1, C'_2, \cdots, C'_s$ are auxiliary vertices created when processing C in Algorithm 4. We have $s = \text{numchild}(C)$. Then we know that the parent of C' in $\mathcal{T}'$ is $C'_{\lfloor (s+i)/2 \rfloor}$. For each $j > 1$, we know that the parent of C'_j in $\mathcal{T}'$ is $C'_{\lfloor j/2 \rfloor}$, and the parent of C'_1 is C. Thus, when we have $\text{rank}(C')$ and $\text{numchild}(C)$, we can compute the path between C' and C in $\mathcal{T}$ locally on a machine. For a leaf-to-root path $C_1, C_2, \cdots, C_m$ in $\mathcal{T}$ stored on a machine, we query the value of $\text{rank}(C_1), \text{numchild}(C_1), \text{rank}(C_2), \text{numchild}(C_2), \cdots, \text{rank}(C_m), \text{numchild}(C_m)$. This corresponds to concurrent read in the CRCW PRAM model ans thus can be simulated in the MPC model in $O(1)$ parallel time and the total space needed does not increase (see e.g., (Goodrich et al., 2011; Andoni et al., 2018)). The algorithm is fully scalable. Then, we are able to compute the corresponding path from C_1 to C_m in $\mathcal{T}'$. Since the length of the path in $\mathcal{T}'$ can increase by a factor at most $\log n$, it is still at most $n^{o(1)}$ and thus it can be stored on a single machine. The total space needed will also blow-up by a factor at most $O(\log n)$. Thus, the total space needed is still $n^{1+o(1)}$. For each leaf-to-root path in $\mathcal{T}'$, we can set the parent of each vertex on the path properly. We use sorting (Theorem 2.2) to remove duplicated parent pointers. Since each vertex must be on at least one leaf-to-root path in $\mathcal{T}'$, the parent of each vertex in $\mathcal{T}'$ is computed.

Now let us consider how to compute $w(C)$ for each vertex C in $\mathcal{T}'$. If C is in $\mathcal{T}$, then the point set C is already stored in the MPC model (see Algorithm 2 and Lemma 4.3). For each auxiliary vertex C' in $\mathcal{T}'$, it corresponds to the point set which is the union of all point sets represented by the leafs in the subtree rooted at C'. Each leaf in $\mathcal{T}'$ corresponds to a singleton. Thus, for each leaf-to-root path in $\mathcal{T}'$ where the leaf is $\{p\}$, we add the point p into every point set represented by a auxiliary vertex C' which is an ancestor of $\{p\}$. This can be done locally since we just need to create a pair $(\text{“}C'\text{”}, p)$. We can use sorting (Theorem 2.2) to remove duplicated points in each C'. For each vertex C in $\mathcal{T}'$, we need to compute $\text{mean}(C)$. We can use sorting (Theorem 2.2) and prefix sum subroutines (Andoni et al., 2018; Goodrich et al., 2011) to compute $\sum_{p \in C} p$ and $|C|$ for each vertex C in $O(1)$ rounds and linear total space, and the algorithm is fully scalable. Thus, we can compute $\text{mean}(C)$ for each vertex C in $\mathcal{T}'$. Then, for each $p \in C$, we can query $\text{mean}(C)$ (corresponding to concurrent read in

CRCW PRAM), and we can use sorting and prefix sum subroutines again to compute $\sum_{p\in C}\mathrm{dist}^2(p,\mathrm{mean}(C))$ for each C in $\mathcal{T}'$ in $O(1)$ parallel time and linear total space. The algorithm is fully scalable.

Therefore, the overall parallel time is $O(1)$ and the total space needed is $n^{1+o(1)}$. Furthermore, the algorithm is fully scalable. □

C.5. Proof of Lemma 4.6

Proof of Lemma 4.6. In the proof, we fix an arbitrary constant $\delta' \in (0,1)$ and suppose that each machine has local memory $\Theta(n^{\delta'}\cdot B)$.

Let us first show that $\mathcal{S}_C$ for each C can be computed in $O(1)$ MPC rounds and $n^{1+o(1)}$ total space. Consider each leaf-to-root path $C_1, C_2, \cdots, C_m$ of $\mathcal{T}'$. For each C_i which is ongoing and each $C_{i'}$ with $i' > i$, we add C_i into $\mathcal{S}_{C_{i'}}$. This can be done locally by creating a pair ("$\mathcal{S}_{C_{i'}}$", "C_i"). According to Lemma 3.7 and Lemma 3.10, the depth of $\mathcal{T}'$ is at most $O(L\log n) = n^{o(1)}$. Thus, the number of pairs that we will create for each leaf-to-root path is at most $O(L^2\log^2 n) = n^{o(1)}$. Notice that if C is a ongoing vertex, then there must be leaf C' such that C and all ancestors of C are on the path from C' to the root, and thus C will be added into $\mathcal{S}_{C''}$ for each C'' which is an ancestor of C. Therefore, all $\mathcal{S}_C$ can be correctly computed. Then we use sorting (Theorem 2.2) to remove duplicates for each $\mathcal{S}_C$. The above steps take $O(1)$ rounds. The total space needed is at most $n^{1+o(1)}$.

Then, we can use sorting (Theorem 2.2) to compute the size of each $\mathcal{S}_C$, and this can be done in $O(1)$ rounds and linear total space (Andoni et al., 2018; Goodrich et al., 2011). We can use sorting again to make each set $\mathcal{S}_C$ with $|\mathcal{S}_C| \le n^{\delta'}$ be stored entirely on a single machine. This takes $O(1)$ rounds and linear total space. Next we show how to compute $\mathrm{DP}(C', i)$ for each $C' \in \mathcal{S}_C$ with $|\mathcal{S}_C| \le n^{\delta'}$. Let us focus on a fixed $\mathcal{S}_C$ with $|\mathcal{S}_C| \le n^{\delta'}$. We know that the set $\mathcal{S}_C$ are entirely stored on a machine. Then for each non-leaf $C' \in \mathcal{S}_C$, and for each finished child C'' of C', we query $\mathrm{DP}(C'', i)$ for all valid i and make them stored on the same machine. Since each vertex can have at most 2 children, the total size of queried outcomes can be at most $2\cdot n^{\delta'}\cdot B$, and thus the outcomes can be stored in the same machine. For each C' in $\mathcal{S}_C$, we further query $w(C')$ and make the queried value stored on the same machine. These steps correspond to the concurrent read operations in the CRCW PRAM model and thus can be simulated in the MPC model in $O(1)$ rounds (Goodrich et al., 2011). Then, we show how to compute $\mathrm{DP}(C', i)$ for each $C' \in \mathcal{S}_C$ and each valid i via local computation. We can prove it by induction over the depth of the subtree rooted at C'. If the depth of the subtree rooted at C' is 0, then C' is a leaf. According to Algorithm 5 and Algorithm 6, $\mathrm{DP}(C', i)$ only depends on $w(C')$. Thus $\mathrm{DP}(C', i)$ can be computed locally. Suppose $\mathrm{DP}(\hat{C}, i)$ can be computed locally for every valid i and every $\hat{C} \in \mathcal{S}_C$ such that the depth of the subtree rooted at $\hat{C}$ is at most $M-1$. Consider $C' \in \mathcal{S}_C$ that the depth of the subtree rooted at C' is M. Notice that $\mathrm{DP}(C', i)$ only depends on the values of $w(C')$ and $\mathrm{DP}(C'', j)$ where C'' can be any child of C' and j can be any valid value. $w(C')$ is already stored on the machine. If C'' is finished, then $\mathrm{DP}(C'', j)$ is also stored on the machine. Otherwise C'' is on going. According to the construction of $\mathcal{S}_C$, since $C' \in \mathcal{S}_C$ and C'' is a ongoing child of C', C'' must be in $\mathcal{S}_C$. By induction hypothesis, $\mathrm{DP}(C'', j)$ can be computed locally. Therefore, in all cases, $\mathrm{DP}(C', i)$ can be computed locally. Thus, each iteration of the loop in Algorithm 7 can be implemented in the MPC model using $O(1)$ rounds and $n^{1+o(1)}$ total space. Now, we analyze the number of iterations the loop can have. Consider an iteration, and let f_1 be the number of ongoing vertices before the iteration and let f_2 be the number of ongoing vertices after the iteration. Let $\mathcal{B} = \{C \mid C \text{ is ongoing after the iteration but all children of } C \text{ are finished after the iteration}\}$. Since the depth of $\mathcal{T}'$ is at most $O(L\log n) = n^{o(1)}$, we have $f_2 \le |\mathcal{B}|\cdot O(L\cdot\log n) = |\mathcal{B}|\cdot n^{o(1)}$. Before the iteration, we know that $|\mathcal{S}_C| > n^{\delta'}$ for every $C \in \mathcal{B}$. Thus, $f_1 \ge |\mathcal{B}|\cdot n^{\delta'}$. Therefore, $f_1/f_2 \ge n^{\Omega(\delta')}$ which implies that the number of ongoing vertices in each iteration will decrease by a factor at least $n^{\Omega(\delta')}$. Thus the number of iterations of the loop in Algorithm 7 is at most $O(1/\delta') = O(1)$.

Finally, let us consider the traverse backwards steps in Algorithm 5 and Algorithm 6. Suppose (C, i) is in $\mathcal{L}$. To determine whether $C \in \mathcal{S}$, we only need to know $w(C)$. For a child C' of C and a valid j, to determine whether (C', j) will be added into $\mathcal{L}$, we only need to know $\mathrm{DP}(C'', j')$ for other child C'' (if exists) of C and all valid j'. Notice that each vertex in $\mathcal{T}'$ has at most 2 children. By induction, for an arbitrary vertex C_1 if all $w(C_1), w(C_2), \cdots, w(C_m)$, all $\mathrm{DP}(C_1,\cdot), \mathrm{DP}(C_2,\cdot), \cdots, \mathrm{DP}(C_m,\cdot)$ and all $\mathrm{DP}(C'_1,\cdot), \mathrm{DP}(C'_2,\cdot), \cdots, \mathrm{DP}(C'_{m-1},\cdot)$ are known, where $C_2, \cdots, C_m$ are all ancestors of C_1 and C'_j is a child (if exists) of C_{j+1} other than C_j, then we can know whether C_1 should be added into $\mathcal{S}$.

Consider a leaf-to-root path $C_1, C_2, \cdots, C_m$. This path is entirely stored on a single machine. We can query all $w(C_1), w(C_2), \cdots, w(C_m)$, all $\mathrm{DP}(C_1,\cdot), \mathrm{DP}(C_2,\cdot), \cdots, \mathrm{DP}(C_m,\cdot)$ and all $\mathrm{DP}(C'_1,\cdot), \mathrm{DP}(C'_2,\cdot), \cdots, \mathrm{DP}(C'_{m-1},\cdot)$ where C'_i is a child of C'_{i+1} other than C_i, and store the queried outcomes on the same machine. This corresponds to the

concurrent read operation in the CRCW PRAM model and thus can be implemented in the MPC model in $O(1)$ rounds and linear total space (Goodrich et al., 2011; Andoni et al., 2018). For each $C \in \mathcal{S}$, since it must be on some leaf-to-root path, it must be correctly added into $\mathcal{S}$. Finally, we can use sorting to remove duplicates in $\mathcal{S}$.

The overall parallel time is $O(1)$ and the total space needed is at most $n^{1+o(1)}$. If $\mathrm{DP}(\cdot,\cdot) \equiv \mathrm{DPcost}(\cdot,\cdot)$, Algorithm 7 exactly simulates Algorithm 5 and thus the output $\mathcal{S}$ is the same as the output of Algorithm 5. If $\mathrm{DP}(\cdot,\cdot) \equiv \mathrm{DPsize}(\cdot,\cdot)$, Algorithm 7 exactly simulates Algorithm 6 and thus the output $\mathcal{S}$ is the same as the output of Algorithm 6. □

C.6. Proof of Theorem 4.8

Proof of Theorem 4.8. According to Lemma 3.2, P satisfies α-center proximity for k-means. Let $\alpha' = (\alpha/4 - 1/2)/1.01$. Then $\alpha' > 1.002$. Let $\mathcal{C}^* = \{C_1^*, C_2^*, \cdots, C_k^*\}$ be the optimal k-means clustering for P. Consider the MPC model that each machine has local memory $\Theta(n^\delta)$ where $\delta \in (0,1)$ is an arbitrary constant. According to Lemma 4.3, we can run Algorithm 2 in the MPC model using $O(1)$ rounds and $n^{1+O(1/\alpha^2)+o(1)}d$ total space. The success probability is at least $1 - 1/\mathrm{poly}(n)$. Let $\mathcal{Q}_0, \mathcal{Q}_1, \cdots, \mathcal{Q}_L$ be the output of Algorithm 2. In the remaining, we condition on that Algorithm 2 succeeds. According to Theorem 3.6, $\forall i \in [k], C_i^* \in \mathcal{Q}_0 \cup \mathcal{Q}_1 \cup \cdots \cup \mathcal{Q}_L$, and $P \in \mathcal{Q}_L, \forall p \in P, \{p\} \in \mathcal{Q}_0$. Then according to Lemma 4.4, we can run Algorithm 3 in the MPC model using $O(1)$ rounds and $n^{1+o(1)}$ total space. Let $\mathcal{T}$ be the output of Algorithm 3. According to Lemma 3.9, $\mathcal{C}^*$ is a pruning set of $\mathcal{T}$ and for any pruning set $\mathcal{S}$ of $\mathcal{T}$ with $|\mathcal{S}| \le k$, we have $\sum_{C\in\mathcal{C}^*}\sum_{p\in C}\mathrm{dist}^2(p,\mathrm{mean}(C)) \le \sum_{C\in\mathcal{S}}\sum_{p\in C}\mathrm{dist}^2(p,\mathrm{mean}(C))$. According to Lemma 4.5, we can run Algorithm 4 in $O(1)$ rounds and $n^{1+o(1)}$ total space. Let $\mathcal{T}'$ be the output of Algorithm 4. For each C in $\mathcal{T}'$, $w(C) = \sum_{p\in C}\mathrm{dist}^2(p,\mathrm{mean}(C))$ is also computed. According to Lemma 3.10, $\mathcal{C}^*$ is a pruning set of $\mathcal{T}'$ and for any pruning set $\mathcal{S}$ of $\mathcal{T}'$ with $|\mathcal{S}| \le k$, we have $\sum_{C\in\mathcal{C}^*} w(C) \le \sum_{C\in\mathcal{S}} w(C)$. According to Lemma 3.7, the depth of $\mathcal{T}$ is at most $O(\log(\text{aspect ratio of } P)) = n^{o(1)}$. According to Lemma 3.10, the depth of $\mathcal{T}'$ is at most $n^{o(1)}$. Thus, we need space $O(\log_{1+\varepsilon/n^{o(1)}}(n\text{aspect ratio of } P)) = n^{o(1)}$ to store $\mathrm{DPsize}(C,0), \mathrm{DPsize}(C,1), \cdots, \mathrm{DPsize}(C,l)$ for each C. According to according to Lemma 4.6, we can use Algorithm 7 to simulate Algorithm 6 in the MPC model using $O(1)$ rounds and $n^{1+o(1)}$ total space. According to Lemma 3.12, we get a pruning set $\mathcal{S}$ of $\mathcal{T}'$ with $|\mathcal{S}| \le k$ such that $\sum_{C\in\mathcal{S}} w(C) \le (1+\varepsilon)\cdot\sum_{C\in\mathcal{C}^*} w(C)$. According to Lemma 3.10, $\mathcal{S}$ is a partition of P and thus $\mathcal{S}$ is an $(1+\varepsilon)$-approximate k-means clustering of P. The overall parallel time is $O(1)$. The total space needed is at most $n^{1+O(1/\alpha^2)+o(1)}d$. □

C.7. Further Improvements

Estimation of the aspect ratio. To estimate $\max_{p,q\in P}\mathrm{dist}(p,q)$, we can choose an arbitrary $x \in P$. Then by triangle inequality, $2\max_{y\in P}\mathrm{dist}(x,y)$ is a 2-approximation of $\max_{p,q\in P}\mathrm{dist}(p,q)$. Now consider how to estimate $\min_{p\neq q\in P}\mathrm{dist}(p,q)$. We choose a random vector $r \in \mathbb{R}^d$ where each entry of r is a standard random Gaussian variable. Thus, $\forall p,q \in P, r^\top(p-q) \sim \mathcal{N}(0,\mathrm{dist}^2(p,q))$. With probability at least $1-1/n^3$, $O(n\cdot\mathrm{dist}(p,q)) \ge |r^\top p - r^\top q| \ge \Omega(\mathrm{dist}(p,q)/n^3)$. By sorting all $r^\top p$, we can find $\min_{p\neq q\in P}|r^\top(p-q)|$. By taking union bound over all pairs $p,q\in P$, with probability at least $1-1/n$, it is at least $\Omega(\min_{p\neq q\in P}\mathrm{dist}(p,q)/n^3)$ and at most $O(n\cdot\min_{p\neq q\in P}\mathrm{dist}(p,q))$. Thus, we are be able to get a $\mathrm{poly}(n)$-approximation of the real aspect ratio. If the real aspect ratio is $2^{n^{o(1)}}$, then the $\mathrm{poly}(n)$-approximate aspect ratio is still $2^{n^{o(1)}}$ and all of our algorithms can work with the approximate aspect ratio. In addition, the above estimation method can be simply implemented in the MPC model using $O(1)$ rounds and linear total space. The algorithm is fully scalable.

Dimension reduction. Let $A \in \mathbb{R}^{d'\times d}$ be a random matrix where each entry is of A is independently drawn from $\mathcal{N}(0,1/d')$. Let d' be a sufficiently large $\Theta(\log n)$. Due to Johnson-Lindenstrauss lemma (Johnson & Lindenstrauss, 1984), with probability at least $1-1/\mathrm{poly}(n)$, $\forall p,q\in P$, $\mathrm{dist}(p,q) \le \mathrm{dist}(1.001Ap, 1.001Aq) \le 1.001^2\,\mathrm{dist}(p,q)$. Let $P' = \{1.001Ap \mid p \in P\}$. We have $P' \subset \mathbb{R}^{d'}$. Suppose P is α-perturbation resilient to k-means. According to Definition 2.1, P' is $\alpha/1.001^2$-perturbation resilient to k-means, and the optimal k-means clustering of P' corresponds to the optimal k-means clustering of P, i.e., $1.001Ap, 1.001Aq$ are in the same optimal cluster for $P' \iff p,q$ are in the same optimal cluster for P. Thus, we only need to solve k-means clustering for P'. Notice that Ap can be easily computed in the MPC model using $O(1)$ rounds and $O(d\cdot d')$ total space, and the algorithm is fully scalable. Thus, we have a fully scalable algorithm which computes P' using $O(1)$ rounds and $O(nd\log n)$ total space. By applying Theorem 4.7 and Theorem 4.8 on P', we obtain Theorem 1.1 and Theorem 1.2.

D. Extension to k-Median

We can extend our techniques to obtain $(1+\varepsilon)$-approximate k-median. In the k-median problem, the goal is to partition P into k clusters $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$ such that $\sum_{i=1}^{k} \sum_{p \in C_i} \mathrm{dist}(p, \mathrm{median}(C_i))$ is minimized, where $\mathrm{median}(C)$ is defined as $\arg\min_{q \in \mathbb{R}^d} \sum_{p \in C} \mathrm{dist}(p, q)$.

It is easy to verify that the structural properties, i.e., Lemma 3.2 and Lemma 3.3 also hold for k-median. Thus, the only modification needed to make our algorithms work for k-median is that we need $w(C) = \sum_{p \in C} \mathrm{dist}(p, \mathrm{median}(C))$ (instead of $w(C) = \sum_{p \in C} \mathrm{dist}^2(p, \mathrm{mean}(C))$) for each vertex C in $\mathcal{T}'$. Next, we show how to estimate $\sum_{p \in C} \mathrm{dist}(p, \mathrm{median}(C))$. We can use coreset to achieve the goal.

Theorem D.1 (see e.g., (Feldman & Langberg, 2011))**.** *Let $\varepsilon \in (0,1)$ be an arbitrary constant. Let $\delta \in (0,1)$. Consider a set $C \subset \mathbb{R}^d$ of m points. There is an MPC algorithm which outputs a set of points C' and weights $w' : C' \to \mathbb{R}_{>0}$ such that with probability at least $1-\delta$, $|C'| \le \mathrm{poly}(d \log(1/\delta)/\varepsilon)$ and*

$$\forall q \in \mathbb{R}^d, (1-\varepsilon) \sum_{p \in C} \mathrm{dist}(p, q) \le \sum_{p \in C'} w'(p) \cdot \mathrm{dist}(p, q) \le (1+\varepsilon) \sum_{p \in C} \mathrm{dist}(p, q).$$

The local memory per machine is $m^{\delta'} \cdot \mathrm{poly}(d \log(1/\delta)/\varepsilon)$ for some arbitrary small constant $\delta' > 0$. The parallel time is $O(1)$, the total space needed is $O(m)$.

As discussed in Section C.7, we can reduce the dimension d to $O(\log n)$. Let δ in the above lemma be $1/n^{10}$. Suppose $\varepsilon \ge 1/n^{o(1)}$. Then we are able to create C' and w' for every C in $\mathcal{T}'$ in $O(1)$ rounds and send C' and w' to a single machine. Thus, we can estimate $w(C)$ up to a $(1+\varepsilon)$ factor by local computation. By taking union bound over all C in $\mathcal{T}'$, the probability that all estimations are good is at least $1 - 1/n^5$. Therefore, we are able to conclude the following theorem:

Theorem D.2. *Let $\alpha > 6.1$ and $\varepsilon \in (1/n^{o(1)}, 0.5)$. Consider a set $P \subset \mathbb{R}^d$ of n points which is α-perturbation resilient to k-median and the aspect ratio of P is at most $2^{n^{o(1)}}$. There is a fully scalable MPC algorithm which outputs the $(1+\varepsilon)$-approximate k-median clustering of P with probability at least $1 - 1/\mathrm{poly}(n)$ using $O(1)$ rounds and $O(nd \log n) + n^{1+O(1/\alpha^2)+o(1)}$ total space.*