Large Scale K-Median Clustering for Stable Clustering Instances

Konstantin Voevodski
Google, Cambridge, USA

Abstract

We study the problem of computing a good k-median clustering in a parallel computing environment. We design an efficient algorithm that gives a constant-factor approximation to the optimal solution for stable clustering instances. The notion of stability that we consider is resilience to perturbations of the distances between the points. Our computational experiments show that our algorithm works well in practice - we are able to find better clusterings than Lloyd's algorithm and a centralized coreset construction using samples of the same size.

1 Introduction

Modern data collections require algorithms that can handle massive amounts of data. In many applications involving online content (for example videos, images, text posts), it is common to have hundreds of millions of data points. Therefore the design and development of large-scale learning algorithms has become a compelling research area. In particular, recently large-scale parallel algorithms have been developed for hierarchical clustering (Bateni et al., 2017), graph partitioning (Aydin et al., 2016), and weighted set cover (Harvey et al., 2018).

Here we study k-median clustering in a parallel computing environment. Given a set of data points X, the k-median clustering problem is to find a set of k centers in X such that the sum of the distances between each data point and the nearest center is minimized. This is a very-well studied problem with several known constant-factor approximations (see Charikar et al. (2002); Jain and Vazirani (2001); Arya et al. (2001)). However, these algorithms are very computationally expensive and do not scale well even to medium-size data sets. Sampling and coreset approaches have been proposed for the k-median problem to support larger data sets, but they are still limited in how much data they can handle. In particular, these approaches still require computing a constant-factor approximation on the sample or coreset, and become infeasible when the size of the required sample or coreset grows too large (see Section 1.2). In addition, online and streaming approaches have been proposed for the k-median problem, but their run time is not feasible for clustering a lot of data points at once (see Section 1.2).

Our objective is to design an algorithm that can easily handle millions of data points. A heuristic way to approach a clustering task of this scale is to first partition the data onto many machines using some application-specific criteria. We can then cluster the data on each machine in parallel. However, this means that the data points assigned to different machines cannot be in the same cluster. If our initial partition strategy is not very good, we then severely limit the quality of the clustering we can output. Another way to approach large-scale clustering is to make strong assumptions about the structure of the data, which can give clustering instances that are easy to solve efficiently even for very large data sets (see Section 3).

In this work we design an efficient k-median clustering algorithm that easily scales to millions of data points. Our algorithm does not use any partitioning heuristics, and makes more realistic assumptions about the structure of the data. The first step of our algorithm samples a set of points $S \subset X$ uniformly at random. We then compute a Voronoi decomposition of X with respect to the points in S. Observe that the Voronoi decomposition may be computed in a distributed fashion in time tn/m, where t is the size of the sample, n is the size of the data set, and m is the number of machines. Given that m can easily be on the order of 10^4, the parallel Voronoi decomposition only takes a few minutes even when n is on the order of 10^{10}. We then compute the clustering by running an in-memory algorithm (on a single machine) that only considers the points in S, whose run time is polynomial in t.

Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130. Copyright 2021 by the author(s).

However, our algorithm is still able to approximate the objective value for the entire data set using the information from the Voronoi cells.

1.1 Our Results

We design and develop a parallel k-median clustering algorithm with strong performance guarantees. In our analysis we prove that if the data has a certain structure, our algorithm outputs a 6-approximation of the optimal solution in expectation. On a single machine our algorithm has a run time of $O(nt + t^3)$, where $t = O(k\log\frac{k}{\delta})$ is the size of the sample used by the algorithm. In a distributed computing environment with m machines, the algorithm can be implemented in time $O(\frac{nt}{m} + t^3)$.

We assume that the clustering instance is resilient to perturbations of the distances between the points - the optimal clustering remains unchanged for small multiplicative perturbations. This assumption has been studied in a variety of previous work (see, in particular Awasthi et al. (2012); Balcan and Liang (2012); Ben-David and Reyzin (2014)). We also assume that the data points are more densely distributed around the centers of the optimal clustering (see Section 2). This is a natural property of a good clustering, and we have observed it to hold for real data.

In Section 4 we show that our algorithm works well in practice. We show that we are able to compute clusterings with better objective value than alternative solutions using a sample of points of the same size. We also show that our center-density assumption holds for the data sets in our study.

1.2 Related Work

There are several constant-factor approximations for k-median. Charikar et al. (2002) give a $6\frac{2}{3}$-approximation algorithm, and Jain and Vazirani (2001) give a 6-approximation. Both approaches use linear programming. Arya et al. (2001) give a $(3 + \epsilon)$-approximation that uses local search. In particular, Arya et al. (2001) present an algorithm that swaps p centers in each stage and gives an approximation ratio of $3 + 2/p$. The most efficient version of this algorithm (swapping one center in each stage) gives a 5-approximation.

In order to handle more data several approaches have been proposed. One approach is to first sample a set of points $S \in X$, and then use one of these constant-factor c-approximation algorithms to cluster S, which still gives a good approximation for X. In particular, for a data set with diameter M in $\mathbb{R}^d$, Mishra et al. (2001) give a sampling algorithm that chooses a sample of size $O((\frac{Mcd}{\epsilon})^2 k)$ and gives a solution where the average distance to the nearest median is within $c \cdot OPT + \epsilon$, where OPT is the value of the optimal solution. Czumaj and Sohler (2004) give an improved algorithm with a sample size of $O(\frac{Mc}{\epsilon^2}kd)$ that gives a solution within $(c + \epsilon)OPT + \epsilon$. Meyerson et al. (2004) present another sampling algorithm for instances with large clusters. It uses the algorithm of Arya et al. (2001) as a subroutine. The resulting approximation ratio is a large constant.

The limitation of these sampling approaches is that we still need to compute a constant-factor approximation on the points in the sample. Such approximations are slow to compute and run on a single machine. Given that these sampling approaches require fairly large sample sizes that may also depend on the data diameter, they become infeasible for very large data sets.

Another approach to handle large data is to use coresets - small subsets of representative points that enable approximating the objective value on the full data set up to a multiplicative factor of $1 \pm \epsilon$. Feldman and Langberg (2011) present a coreset of size $t = O(\frac{kd}{\epsilon^2})$ that runs in time $O(ndk + \log^2 n + k^2 + t \log n)$, where n is the number of data points and d is the dimension of the data. Their algorithm takes a k-median solution as input, and samples points based on their contribution to the objective in this solution. However, the analysis requires first computing a constant-factor approximation for the entire data set, which is infeasible for large data sets. In Section 4 we compare the performance of our algorithm with the algorithm of Feldman and Langberg (2011). In our implementation of the construction of Feldman and Langberg (2011) we use a heuristic to compute the initial k-median solution.

Balcan et al. (2013) improve on the limitations of Feldman and Langberg (2011) by showing how to construct the coreset in a distributed fashion on m machines, where each machine computes a constant-factor approximation on a subset of the data of size n/m. The size of the resulting coreset is $O(\frac{kd}{\epsilon^2} + mk)$. However, when the number of machines is large (say $m = 10^4$), this still gives a coreset that is too large, given that we must compute a constant-factor approximation on the points in the coreset in order to preserve the approximation ratio on the full data set. On the other hand, choosing a smaller m is problematic because then n/m (the size of each subset) becomes too large, and we will not be able to compute a constant-factor approximation on each subset.

The perturbation resilience property considered here was also studied by Awasthi et al. (2012) and Balcan and Liang (2012). Awasthi et al. (2012) give a polynomial-time algorithm for α-perturbation resilient instances for any center-based clustering objective in a metric space. Their algorithm works for $\alpha \geq 3$ when the centers are restricted to only the data points. Balcan and Liang (2012) improve this bound to $\alpha \geq 1+\sqrt{2}$ using a more sophisticated algorithm, and also study a noisy version of the problem where only a subset of the data points satisfy α-perturbation resilience.

The algorithms of Awasthi et al. (2012) and Balcan and Liang (2012) compute a hierarchical clustering of the data points and then use dynamic programming to find the best k-pruning of the hierarchical clustering tree. Given the stability assumptions, they return the optimal clustering, but their run time is infeasible for large data sets. It takes $O(n^3)$ time to compute a hierarchical clustering, where n is the number of data points. Computing the dynamic programming solution also requires $O(n^2)$ time just to compute the cost of a single cluster for each sub-tree.

The complexity of perturbation resilience was studied by Ben-David and Reyzin (2014). They prove that finding the optimal k-median clustering is NP-hard if the data satisfies α-perturbation resilience for $\alpha < 2$. On the other hand, they show that if $\alpha > 2+\sqrt{3}$, then the optimal clustering satisfies *strict separation* (see Definition 2.7). When this is the case, Balcan et al. (2008) show that we can construct a hierarchical clustering of the data using single-linkage, and some pruning of this tree must be equivalent to the optimal clustering. As before, this pruning may be computed using dynamic programming. However, this algorithm is still not feasible for large-scale data.

Other MapReduce, online, and streaming algorithms have been proposed for the k-median problem. The MapReduce algorithm from Ene et al. (2011) uses a weighted k-median c-approximation algorithm as a subroutine, and then outputs a $(10c+3)$-approximation to the optimal solution. Using the $(3+\epsilon)$-approximation algorithm from Arya et al. (2001) as a subroutine, this construction at best gives a 33-approximation. The online algorithm from Lattanzi and Vassilvitskii (2017) optimizes for clustering consistency in the online setting, where points $x_1, x_2, \dots x_n$ arrive one at a time, rather than the overall run time for clustering n points. In fact, their construction requires calling a constant-factor approximation algorithm for k-median n times. Similarly, the streaming algorithm for k-median from Charikar et al. (2003) optimizes for memory efficiency, rather than the overall run time. Its construction requires solving a facility-location problem in each phase, where the number of phases is bounded by the total number of points. While requiring limited memory, the overall run time is not feasible for large-scale data.

Large-scale algorithms have also been studied in the context of k-center clustering (Malkomes et al., 2015), Gaussian mixture models (Lucic et al., 2018), and k-means clustering (Bachem et al., 2018). The coreset for Gaussian mixture models from Lucic et al. (2018) uses k-means++ as a subroutine, and the overall construction is very similar to the centralized coreset construction of Feldman and Langberg (2011) for k-median, which is one of our experimental baselines in Section 4.

2 Preliminaries

Suppose we are given a metric space (X, d), where $d : X \times X \rightarrow R_{\geq 0}$, and $n = |X|$. The k-median objective function is to find k centers $c_1, \dots c_k \in X$ that minimize the sum of the distances of each point to the nearest center: $\sum_{x \in X} \min_i d(x, c_i)$. We use $C = C_1, \dots C_k$ to refer to the corresponding clusters (where each point is assigned to the nearest center).

We use $c_1^*, \dots c_k^*$ to refer to the centers that optimize the k-median objective, and use $C^* = C_1^*, \dots C_k^*$ to refer to the corresponding clustering. We use OPT to refer to the optimal objective value.

We assume that our clustering instance is stable with respect to multiplicative perturbations of the distances between the points. We next formally define this property.

Definition 2.1. *Given a metric space (X, d) and a parameter $\alpha > 1$, an α-perturbation of d is a function $d' : X \times X \rightarrow R_{\geq 0}$, such that for any pair of points $x, y \in X$, we have $d(x, y) \leq d'(x, y) \leq \alpha d(x, y)$. We say that a clustering instance satisfies α-perturbation stability for objective function Φ if the unique optimal clustering for (X, d) under Φ is the same as the unique optimal clustering for (X, d') under Φ, where d' is any α-perturbation of d.*

We next state two properties that follow from α-perturbation stability, which are used throughout our analysis. These properties are proved in Awasthi et al. (2012).

Proposition 2.2. *Suppose a clustering instance satisfies α-perturbation stability for the k-median objective function. Then for any $x \in C_i^*$ and any other cluster $C_{j \neq i}^*$, we must have $d(x, c_j^*) > \alpha d(x, c_i^*)$.*

Proposition 2.3. *Suppose a clustering instance satisfies α-perturbation stability for the k-median objective function. Then for any pair of points $x \in C_i^*$ and $y \in C_{j \neq i}^*$, we must have $d(x, y) > (\alpha - 1) d(x, c_i^*)$.*

In our analysis we use r_i to denote the *radius* of cluster C_i, which is defined as the maximum distance between c_i and any other point in C_i: $r_i = \max_{x \in C_i} d(x, c_i)$. Similarly, we use r_i^* to denote the radius of cluster C_i^*. We use $B(x, d)$ to denote the ball of radius d around x: $B(x, d) = \{y \in X : d(x, y) \leq d\}$.

We observe that a lot of the points in C_i^* may lie close to the center c_i^*. We formalize this observation with the following definition.

Definition 2.4. *We say that a k-median clustering $C = C_1, \dots C_k$ satisfies γ-center density if there is a constant $0 < c < 1$, and a constant $0 < \gamma < 1$, such that for each $C_i \in C$, we have $|B(c_i, \gamma r_i)| \geq c|C_i|$. We say that a k-median clustering instance satisfies γ-center density if the optimal clustering C^* satisfies this property.*

Our algorithm computes a hierarchical clustering tree using *minimax* distance, which is defined as follows.

Definition 2.5. *For two sets of points $S_1, S_2 \subset X$, we use $d_m(S_1, S_2)$ to denote the minimax distance between S_1 and S_2, which is defined as $d_m(S_1, S_2) = \min_{x \in S_1 \cup S_2} \max_{y \in S_1 \cup S_2} d(x, y)$.*

In our analysis we show that the hierarchical clustering computed by our algorithm is *consistent* with the optimal clustering C^*. Our consistency property is defined as follows.

Definition 2.6. *Let T be a hierarchical clustering tree of a set of points $S \subseteq X$, and let the clustering C be defined as C^* restricted to the points in S: $C_i = C_i^* \cap S$. We say that T is consistent with C^* if for each node N in T, and each cluster $C_i \in C$, we either have $N \cap C_i = \emptyset$, $N \subseteq C_i$, or $C_i \subseteq N$.*

Note that for each $C_i^* \in C^*$, this property requires that all points from C_i^* must be merged before merging them with any points from $C_{j \neq i}^*$.

We discuss instances when C^* satisfies additional structural properties. In particular, we consider the *strict separation* and *strict threshold separation* properties of a clustering, which are defined as follows.

Definition 2.7. *We say that a clustering C satisfies strict separation if for all $x, y \in C_i$ and $z \in C_{j \neq i}$ we have $d(x, y) < d(x, z)$.*

Definition 2.8. *We say that a clustering C satisfies strict threshold separation if there exists a threshold t, such that for all $x, y \in C_i$ we have $d(x, y) \leq t$, and for all $x \in C_i$ and $y \in C_{j \neq i}$ we have $d(x, y) > t$.*

3 Large Scale K-Median Clustering

In order to motivate the design of our algorithm, we observe that the difficulty of the problem may vary depending on the structure of the optimal clustering. In particular, if the optimal clustering has very strong structure, such as strict threshold separation (see Definition 2.8), then we can recover it with a very simple and efficient algorithm that runs on a single machine. This observation is formalized in Proposition 3.1.

Proposition 3.1. *Suppose a k-clustering C for (X, d) satisfies strict threshold separation and each cluster in C has size $\Omega(n/k)$. Then there exists an algorithm that outputs C with probability at least $1 - \delta$ and runs on a single machine in time $O(k^2 log\frac{k}{\delta} + kn)$.*

Proof. It suffices for the algorithm to sample $t = O(k\log\frac{k}{\delta})$ points from X uniformly at random. Let S refer to the sampled points. We can verify that with probability at least $1 - \delta$, S contains a point from each cluster in C. When this is the case, it suffices to run k iterations of farthest-first traversal on the points in S - in each iteration we select the point in S that has the largest min-distance to the points we already selected. The first point in the traversal may be selected in any manner. This traversal may be implemented in $O(kt)$ time. We then output the clustering corresponding to the k centers output by the traversal (by assigning each point to the nearest center). This requires kn time. We can verify that if C satisfies strict threshold separation, the output clustering must be equivalent to C. ☐

Note that Proposition 3.1 applies to any k-clustering, and clearly also applies to the optimal k-median clustering if it satisfies these structural assumptions. However, such assumptions are very strong and are less likely to hold in practice. We next present our algorithm that works under more realistic assumptions about the data. Like the algorithm described in the proof of Proposition 3.1, our algorithm is still able to utilize a uniform sample of the data points. However, we need more information about each sampled point, which we can compute in parallel, and a more sophisticated procedure to output a clustering that approximates the optimal solution.

3.1 Algorithm Description

Our algorithm first selects a subset of *seed* points $S \subset X$ uniformly at random. We then compute a Voronoi decomposition of X with respect to the points in S. For each $s \in S$, let $v(s)$ denote the points in its Voronoi cell, and let $c(s)$ denote the *cost* of s, which is defined as $c(s) = \sum_{x \in v(s)} d(x, s)$. We compute $|v(s)|$ and $c(s)$ for each seed point $s \in S$. This information is used to approximate the objective value of the clusterings considered by our algorithm.

Our algorithm then computes a hierarchical clustering of S in a bottom-up fashion using *minimax distance* (see Definition 2.5). In each iteration, two nodes with the smallest minimax distance are merged, until only one node is left. Note that if we start with $t = |S|$ points, we will have $2t - 1$ nodes in total. We will use $N_1, N_2, \dots N_{2t-1}$ to refer to these nodes indexed in depth-first order.

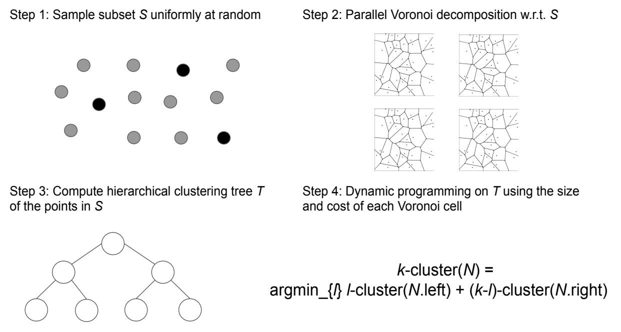

Step 1: Sample subset S uniformly at random
Step 2: Parallel Voronoi decomposition w.r.t. S
Step 3: Compute hierarchical clustering tree T of the points in S
Step 4: Dynamic programming on T using the size and cost of each Voronoi cell
k-cluster(N) = argmin_{l} l-cluster(N.left) + (k-l)-cluster(N.right)

Figure 1: A high-level description of the algorithm components.

We then apply dynamic programming to find the best k-pruning of the hierarchical clustering tree. Our dynamic programming only considers clusterings of the seed points, but approximates the objective value for the entire data set using the information from the Voronoi cell of each seed. Figure 1 gives a high-level description of the components of our algorithm, and we follow with a more exact description and its analysis.

We will use s_i to refer to the seed points in node N_i. In our analysis we will also use $p_i = \{x \in X : x \in v(s)$ and $s \in s_i\}$ to refer to the points in the corresponding Voronoi cells. In order to define the table of solutions constructed by the dynamic programming, we will use $\text{cost}(i, j)$ to refer to the cost of clustering N_i using j clusters.

Algorithm 1 Large-Scale-K-Median(X, d, k, δ)

```
• Let $t = O(k\log\frac{k}{\delta})$
• Sample $t$ points $S \subset X$ uniformly at random
• Compute a Voronoi decomposition of $X$ w.r.t. $S$
• For each $s \in S$, let $v(s)$ denote the points in its Voronoi cell, let $c(s) = \sum_{x \in v(s)} d(x, s)$
• Let $T$ be the hierarchical clustering tree of $S$ computed using minimax distance
• Run Dynamic-Programming($T, k$)
• Let $N_r$ be the root of $T$
• Output clustering corresponding to $cost(r, k)$
```

Algorithm 2 Dynamic-Programming(T, k)

```
Let $N_1, N_2, \ldots N_{2t-1}$ be a depth-first traversal of $T$
for $i = 1$ to $2t - 1$ do
    for $j = 1$ to $k$ do
        Let $cost(i, j) = $ Cluster($N_i, j$)
    end for
end for
```

Algorithm 3 Cluster(N_i, j)

```
if $j = 1$ then
    Let $s_i$ be the seed points in node $N_i$
    return $\min_{x \in s_i}$ One-Clustering-Cost($x, s_i$)
end if
Let $i_1$ be the index of the left child of $N_i$
Let $i_2$ be the index of the right child of $N_i$
return $\min_{0<l<j} cost(i_1, l) + cost(i_2, j - l)$
```

Algorithm 4 One-Clustering-Cost(x, s_i)

```
Initialize cost = 0
for $s \in s_i$ do
    cost $\leftarrow$ cost $+ c(s) + d(x, s) \cdot |v(s)|$
end for
return cost
```

The algorithm is described in more detail in Algorithm 1. For simplicity, we omit the details regarding the initialization of the table for leaf nodes and nodes with less than two children. In order to output the best clustering, it suffices to store the centers corresponding to the clusterings in the table of solutions (these details are also omitted).

3.2 Performance Analysis

We state the theorem regarding the performance of our algorithm in Theorem 3.2.

Theorem 3.2. *Suppose the k-median clustering for (X, d) satisfies α-perturbation stability and γ-center density for $\alpha \geq 3 + 2\gamma$. Suppose each cluster in the optimal clustering has size $\Omega(n/k)$. Then with probability at least $1 - \delta$, Algorithm 1 outputs a clustering with expected objective value at most $6 \cdot OPT$, and runs in time $O(nt + t^3)$, where $t = O(klog\frac{k}{\delta})$ is the size of the sample used by the algorithm. In a parallel computing environment with m machines, Algorithm 1 has a run time of $O(\frac{nt}{m} + t^3)$.*

In order to prove Theorem 3.2 we next state and prove several observations regarding the structure of the clustering instance, and the performance of the individual components of our algorithm. Lemma 3.3 shows that given our perturbation stability assumption, any pair of points $x \in C_i^*$ and $y \in C_j^*$ must be well-separated with respect to the radius of C_i^*.

Lemma 3.3. *Suppose the k-median clustering for (X, d) satisfies α-perturbation stability for $\alpha \geq 3 + 2\gamma$. For any pair of points $x \in C_i^*$ and $y \in C_{j \neq i}^*$, we must have $d(x, y) > (1 + \gamma)r_i^*$.*

Proof. Let z be the point in C_i^* that is farthest from c_i^* (ties broken arbitrarily). In other words, we have $d(z, c_i^*) = r_i^*$. We show that if $d(x, y) \leq (1 + \gamma)r_i^*$, then z is too close to c_j^*, violating our perturbation stability assumption. To see this, observe that by the triangle inequality we have $d(z, c_j^*) \leq d(z, c_i^*) + d(c_i^*, x) + d(x, y) + d(y, c_j^*) = r_i^* + d(c_i^*, x) + d(x, y) + d(y, c_j^*)$. Then by Proposition 2.3, given that $\alpha \geq 3$, we must have $d(x, c_i^*) < d(x, y)/(\alpha - 1) < d(x, y)/2$, and similarly $d(y, c_j^*) < d(x, y)/(\alpha - 1) < d(x, y)/2$. Then if we have $d(x, y) \leq (1 + \gamma)r_i^*$, we must have $d(z, c_j^*) < (3 + 2\gamma)r_i^*$, violating perturbation stability for z per Proposition 2.2. □

Continuing our analysis, let T_S be the hierarchical clustering tree of the seed points S that is computed by our algorithm. Let T_X be the corresponding hierarchical clustering tree of X, which is given by the Voronoi cells of the seed points. In other words, each node N_i in T_S corresponds to a node N_i' in T_X, which contains the points p_i. Note that we only use T_X in our analysis; our algorithm does not compute it. Lemma 3.4 shows that given our assumptions, both T_S and T_X must be consistent with C^*, per our definition of consistency in Definition 2.6.

Lemma 3.4. *Suppose the k-median clustering for (X, d) satisfies α-perturbation stability and γ-center density for $\alpha \geq 3 + 2\gamma$. Suppose each cluster in the optimal clustering has size $\Omega(n/k)$. Let T_S be the hierarchical clustering tree of the seed points S computed by our algorithm. Let T_X be the corresponding hierarchical clustering tree of X. Then with probability at least $1-\delta$, T_S and T_X are consistent with the optimal clustering C^*.*

Proof. For each cluster C_i^*, let us use s_i^* to refer to the closest seed point to c_i^* (ties broken arbitrarily). If our clustering instance satisfies γ-center density, and each cluster in the optimal clustering has size $\Omega(n/k)$, we can verify that if we select $O(klog\frac{k}{\delta})$ seed points, with probability at least $1 - \delta$, for each cluster C_i^*, we will have $d(s_i^*, c_i^*) \leq \gamma r_i^*$.

Whenever this is the case, we prove that T_S must be consistent with C^* by induction on the construction of T_S. At the start, for each cluster A we clearly have $A \subseteq C_i^*$ for some $C_i^* \in C^*$. We now verify that if $A \subset C_i^*$, and $A' \subset C_{j\neq i}^*$ or A' is the union of two or more other clusters in C^*, there must be another cluster $B \subset C_i^*$ such that $d_m(A, B) < d_m(A, A')$.

To verify this, let us consider the cluster assignment of s_i^*. If $s_i^* \in A$, then we may choose B to be any other cluster that is a strict subset of C_i^*. Otherwise, we choose B to be the cluster that contains s_i^*. Using the triangle inequality, we can verify that $d_m(A, B) \leq (1 + \gamma)r_i^*$. On the other hand, Lemma 3.3 shows that the distance between any pair of points $x \in A$ and $y \in A'$ must be more than $(1 + \gamma)r_i^*$, which implies that $d_m(A, A') > (1 + \gamma)r_i^*$.

Finally, we prove that for any point $x \in C_i^*$ in the Voronoi cell of a seed point $s \in S$, we must also have $s \in C_i^*$. To see this, consider that by Lemma 3.3, the distance between x and any seed point $s' \in C_{j\neq i}^*$ must satisfy $d(x, s') > (1 + \gamma)r_i^*$. On the other hand, the distance between x and s_i^* must satisfy $d(x, s_i^*) \leq d(x, c_i^*) + d(c_i^*, s_i^*) \leq r_i^* + \gamma r_i^* = (1 + \gamma)r_i^*$. Clearly, x will then be in the Voronoi cell of s_i^*, or another seed point from C_i^*. It follows that whenever T_S is consistent with C^*, T_X must be consistent with C^* as well. □

We next state and prove a lemma regarding the selection of the seed points. Lemma 3.5 shows that in expectation they give a 2-approximation with respect to the cost of the optimal centers.

Lemma 3.5. *Let C_i be a cluster with center c_i, and let x be a point in C_i chosen uniformly at random. Let us define $cost(y) = \sum_{z\in C_i} d(z, y)$. Then we have $\mathbb{E}[cost(x)] \leq 2 \cdot cost(c_i)$.*

Proof. By the triangle inequality, we have $cost(x) \leq \sum_{z\in C_i}(d(z, c_i) + d(c_i, x)) = \sum_{z\in C_i} d(z, c_i) + |C_i|d(c_i, x) = cost(c_i) + |C_i|d(c_i, x)$. Also, observe that for x chosen uniformly at random in C_i, we have $\mathbb{E}[d(c_i, x)] = (1/|C_i|) \cdot \sum_{z\in C_i} d(z, c_i) = (1/|C_i|) \cdot cost(c_i)$. Combining these two observations, by linearity of expectation we have $\mathbb{E}[cost(x)] \leq 2 \cdot cost(c_i)$. □

Finally, Lemma 3.6 shows that the cost of the center computed by Algorithm 4 on only the seed points is a 3-approximation with respect to the cost computed on the entire data set.

Lemma 3.6. *For node N_i, let s_i refer to the seed points in N_i, and let $p_i = \{x \in X : x \in v(s)$ and $s \in s_i\}$ refer to the points in the corresponding Voronoi cells. For any $x \in s_i$, let $\tilde{cost}(x)$ be the output of One-Clustering-Cost(x, s_i), and let $cost(x) = \sum_{y\in p_i} d(y, x)$. Then we must have $cost(x) \leq \tilde{cost}(x) \leq 3 \cdot cost(x)$.*

Proof. For any point $y \in p_i$, let $s(y)$ be the seed point $s \in s_i$ such that y is in the Voronoi cell of s. Observe that by construction we have $\tilde{cost}(x) = \sum_{y\in p_i} d(y, s(y)) + d(s(y), x)$.

Clearly, for y in the Voronoi cell of x, we have $d(y, s(y)) + d(s(y), x) = d(y, x) + d(x, x) = d(y, x)$. It suffices to show that for points $y \in p_i$ in other Voronoi cells, we have $d(y, x) \leq d(y, s(y)) + d(s(y), x) \leq 3d(y, x)$.

To verify the first part, by the triangle inequality, we have $d(y, x) \leq d(y, s(y)) + d(s(y), x)$. To verify the second part, observe that by the triangle inequality we have $d(x, s(y)) \leq d(x, y) + d(y, s(y))$. Adding $d(y, s(y))$ to both sides, we have $d(x, s(y)) + d(y, s(y)) \leq d(x, y) + 2d(y, s(y))$. Also observe that given that y is in the Voronoi cell of $s(y)$, we have $d(y, s(y)) \leq d(y, x)$. Combining the last two inequalities, we have $d(y, s(y)) + d(s(y), x) \leq 3d(y, x)$, as required. □

The observations in Lemma 3.4, Lemma 3.5, and Lemma 3.6 enable us to prove Theorem 3.2. We now follow with the proof of Theorem 3.2.

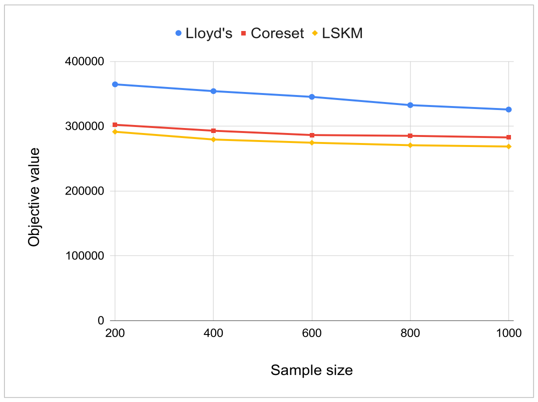
Proof of Theorem 3.2. By Lemma 3.4, there must be a pruning of T_X that is equivalent to C^*. Consider any $C_i^* \in C^*$. For any point $x \in C_i^*$ define $cost(x) = \sum_{y \in C_i^*} d(y, x)$. By Lemma 3.5, for each seed point $s \in C_i^*$, in expectation we have $cost(s) \leq 2 \cdot cost(c_i^*)$. Then by Lemma 3.6, we have $cost(s) \leq \tilde{cost}(s) \leq 3 \cdot cost(s)$, where $\tilde{cost}(s)$ is the cost of s that is computed by our algorithm. Combining these observations, we see that in expectation we have $\tilde{cost}(s) \leq 6 \cdot cost(c_i^*)$. Therefore in expectation the objective value for C_i^* that is computed by our algorithm must be within $6 \cdot cost(c_i^*)$. Considering that this statement holds for each cluster in C^*, in expectation the objective value of this pruning of the hierarchical clustering tree (which is equivalent to C^*) must be within $6 \cdot OPT$. Also, observe that by Lemma 3.6, the cost of a clustering that is computed by our algorithm (using only the seed points) is an upper-bound on its actual cost. Clearly, the dynamic programming solution will then return either the pruning that is equivalent to C^* or a different pruning with objective value that is within $6 \cdot OPT$ in expectation.

To verify the run time, the Voronoi decomposition can be computed in $O(nt)$ time on a single machine, given that it requires t comparisons for each of the n points. In a parallel computing environment, we may partition the n points into m equal-sized sets, assign each set to a different machine, and then compute the Voronoi decomposition for each set in parallel, reducing the run time by a factor of m. In particular, in a parallel implementation, the t seed points are passed to each machine; each machine then computes the quantities $|v(s)|$ and $c(s)$ for the points assigned to it, which are then added together to compute these quantities for the entire data set.

It takes $O(t^3)$ time to compute the hierarchical clustering tree of the t seed points. The dynamic programming table is then constructed iteratively for each of the $2t - 1$ nodes in the tree. For each node, it takes $O(t^2)$ time to compute the one-clustering cost, and $O(k^2)$ to compute the j-clustering cost for $2 \leq j \leq k$. Given that we choose a setting of t that is larger than k, the dynamic programming table is then computed in $O(t^3)$ time. □

4 Experiments

We evaluate the performance of our algorithm on the Covertype data set from the UCI Machine Learning Repository (Dheeru and Taniskidou, 2017), as well as the Quantum Physics and Protein Homology data sets from KDD Cup. Each data set contains n points in $\mathbb{R}^d$. For the Covertype data set we have $n = 581012$ and $d = 54$. For the Quantum Physics data set we have $n = 50000$ and $d = 78$. For the Protein Homology data set we have $n = 145751$ and $d = 74$. We normalize the data for each dimension such that all entries are in [0,1] (where 0 corresponds to the minimum value and 1 corresponds to the maximum value). For the Quantum Physics data set we also discard the 8 dimensions that have missing data (see KDD Cup).



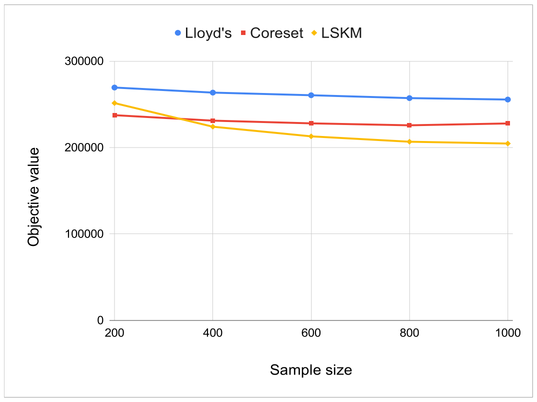


Figure 2: Performance comparison for the Covertype data set. Top: results for $k = 50$. Bottom: results for $k = 100$.

Our Algorithm 1 is termed *Large Scale K-Median (LSKM)*. We compare performance with two alternative algorithms. The first is simply Lloyd's algorithm run on a sample of points chosen uniformly at random, which we term *Lloyd's*. The initial centers for Lloyd's are

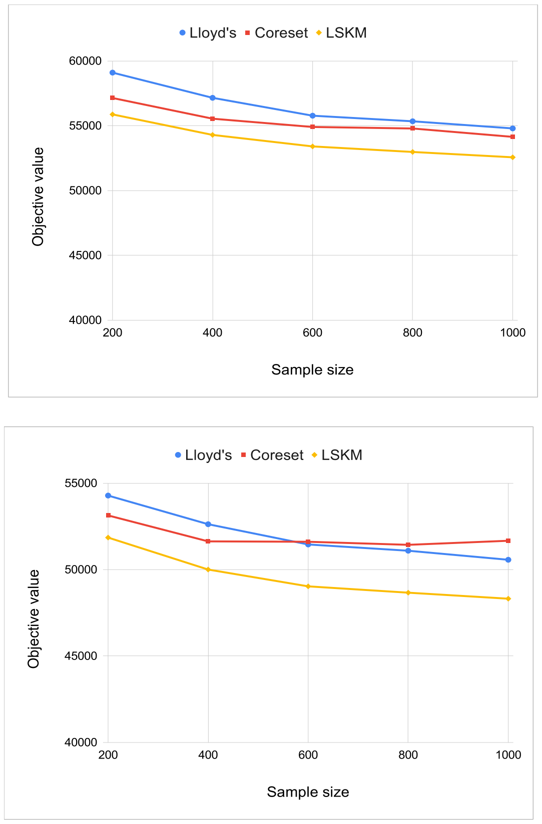


Figure 3: Performance comparison for the Quantum Physics data set. Top: results for $k = 50$. Bottom: results for $k = 100$.

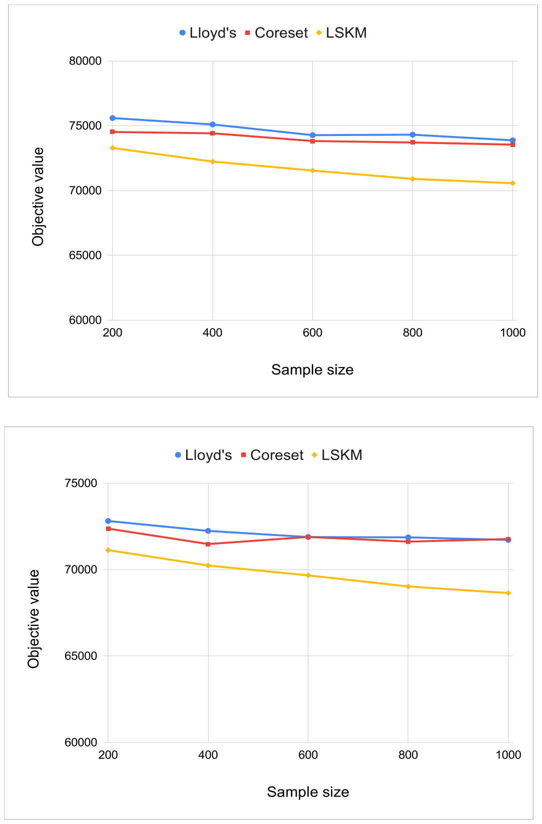


Figure 4: Performance comparison for the Protein Homology data set. Top: results for $k = 50$. Bottom: results for $k = 100$.

chosen uniformly at random. The second algorithm is the centralized coreset construction from Feldman and Langberg (2011). Their construction takes a k-median solution as input, and computes the coreset by sampling points based on their contribution to the objective in this solution. We compute the initial k-median solution using *Lloyd's*. We then use the algorithm of Feldman and Langberg (2011) to compute the coreset. Finally, we compute the clustering of the coreset using the algorithm of Arya et al. (2001), swapping one center per stage. We refer to the entire procedure as *Coreset*.

We run each algorithm using a sample of points and/or coreset of the same size, and record the objective value of the output clustering (specified by a set of centers). Note that the objective value is computed with respect to the entire data set. We rerun each algorithm 10 times and report the average objective value. We also vary the number of clusters (k), and report the results for $k = 50$ and $k = 100$. Figure 2 displays the experimental results for the Covertype data set. Figure 3 and Figure 4 display the results for the Quantum Physics and Protein Homology data sets, respectively.

4.1 Evaluation of Center Density

We observe that our γ-center density property holds for the clusterings in our study for fairly low settings of γ (see Definition 2.4). In particular, for each data set we take the best k-median clustering computed by our algorithm (the one with the lowest objective value), and consider the distances of the points in each cluster to the cluster center. For $k = 100$ we find that γ-center density holds for $c = 0.1$ and $\gamma = 0.29, 0.59, 0.62$ for the Covertype, Quantum Physics, and Protein Homology data sets, respectively. For $k = 100$ we find that the property holds for $c = 0.1$ and $\gamma = 0.32, 0.70, 0.75$.

Assuming that the optimal clustering C^* is structurally similar to our best clustering, these conditions improve the bound on α in our theoretic analysis (see Theorem 3.2).

5 Discussion

We design a parallel k-median clustering algorithm that easily scales to very large data sets. We prove that if the data has a certain structure, our algorithm computes a 6-approximation of the optimal objective value in expectation. Our experiments show that we are able to compute much better clusterings than alternative methods using samples of the same size.

It may be possible to further relax our assumptions on the structure of the data. In particular, our analysis requires that the clustering instance satisfy α-perturbation stability for $\alpha > 3$. It is NP-hard to find the optimal clustering if the data satisfies this property for $\alpha < 2$ (Ben-David and Reyzin, 2014). At the same time, Balcan and Liang (2012) give an algorithm that finds the optimal clustering for $\alpha \geq 1+\sqrt{2} \approx 2.41$. This algorithm does not scale well, but these results show that it may be possible to further relax the assumption on α for a parallel clustering algorithm.

References

V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. In *STOC*, pages 21–29, 2001.

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

K. Aydin, M. Bateni, and V. Mirrokni. Distributed balanced partitioning via linear embedding. In *WSDM*, pages 387–396, 2016.

O. Bachem, M. Lucic, and A. Krause. Scalable k-means clustering via lightweight coresets. In *KDD*, pages 1119–1127, 2018.

M. F. Balcan and Y. Liang. Clustering under perturbation resilience. In *ICALP*, pages 63–74, 2012.

M. F. Balcan, A. Blum, and S. Vempala. A discriminative framework for clustering via similarity functions. In *STOC*, pages 671–680, 2008.

M. F. Balcan, S. Ehrlich, and Y. Liang. Distributed k-means and k-median clustering on general topologies. In *NIPS*, pages 1995–2003, 2013.

M. Bateni, S. Behnezhad, M. Derakhshan, M. Hajiaghayi, R. Kiveris, S. Lattanzi, and V. Mirrokni. Affinity clustering: Hierarchical clustering at scale. In *NIPS*, pages 6867–6877, 2017.

S. Ben-David and L. Reyzin. Data stability in clustering: A closer look. *Theoretical Computer Science*, 558:51–61, 2014.

M. Charikar, S. Guha, Éva Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *Journal of Computer and System Sciences*, 65(1):129–149, 2002.

M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In *STOC*, pages 30–39, 2003.

A. Czumaj and C. Sohler. Sublinear-time approximation for clustering via random sampling. In *ICALP*, pages 396–407, 2004.

D. Dheeru and E. K. Taniskidou. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

A. Ene, S. Im, and B. Moseley. Fast clustering using mapreduce. In *KDD*, pages 681–689, 2011.

D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *STOC*, pages 569–578, 2011.

N. J. A. Harvey, C. Liaw, and P. Liu. Greedy and local ratio algorithms in the mapreduce model. In *SPAA*, pages 43–52, 2018.

K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.

KDD Cup. Kdd cup 2004: Particle physics; plus protein homology prediction. `https://www.kdd.org/kdd-cup/view/kdd-cup-2004/Data`, 2004.

S. Lattanzi and S. Vassilvitskii. Consistent k-clustering. In *ICML*, pages 1975–1984, 2017.

M. Lucic, M. Faulkner, A. Krause, and D. Feldman. Training gaussian mixture models at scale via coresets. *Journal of Machine Learning Research*, 18(160): 1–25, 2018.

G. Malkomes, M. J. Kusner, W. Chen, K. Q. Weinberger, and B. Moseley. Fast distributed k-center clustering with outliers on massive data. In *NIPS*, pages 1063–1071, 2015.

A. Meyerson, L. O'Callaghan, and S. Plotkin. A k-median algorithm with running time independent of data size. *Machine Learning*, 56(1):61–87, 2004.

N. Mishra, D. Oblinger, and L. Pitt. Sublinear time approximate clustering. In *SODA*, pages 439–447, 2001.