Extreme k-Center Clustering

MohammadHossein Bateni,[1] Hossein Esfandiari,[1] Manuela Fischer,[2] Vahab Mirrokni[1]
[1]Google Research, NYC, New York, USA
[2]ETH, Zurich, Switzerland
bateni@google.com, esfandiari@google.com, manuela.fischer@inf.ethz.ch, mirrokni@google.com

Abstract

Metric clustering is a fundamental primitive in machine learning with several applications for mining massive datasets. An important example of metric clustering is the k-center problem. While this problem has been extensively studied in distributed settings, all previous algorithms use $\Omega(k)$ space per machine and $\Omega(nk)$ total work. In this paper, we develop the first highly scalable approximation algorithm for k-center clustering, with $\widetilde{O}(n^{\varepsilon})$ space per machine and $\widetilde{O}(n^{1+\epsilon})$ total work, for arbitrary small constant ε. It produces an $O(\log\log\log n)$-approximate solution with $k(1+o(1))$ centers in $O(\log\log n)$ rounds of computation.

1 Introduction

Designing scalable algorithms has become increasingly critical in the new era of massive datasets. Many of the classic efficient algorithms developed over decades are not effective anymore in handling very large datasets. For example, the simplest sequential greedy algorithms fail to work when they need to make millions of iterations over billions of data points. It is inevitable to resort to distributed algorithms. These, however, often come with a slightly worse solution quality. The design of scalable algorithms is all about balancing the feasibility of running the algorithm versus the accuracy of the solution. The two main dimensions to scale an algorithm are to improve its total work, or to distribute the work among several machines.

In this paper, we develop *scalable approximation algorithms* for metric clustering, where the high-level goal is to identify k groups for given data points based on their proximity in a metric space. The *k-center* problem is an important and well-studied formulation for metric clustering (Gonzalez 1985).

Problem Statement: In this problem, we are given a parameter $k \geq 1$ and a set V of n points in a metric space. The goal is to find a subset $\mathcal{S} \subseteq V$ of size k, called centers, such that the maximum distance of any point in V to $\mathcal{S}$ is minimized. This maximum distance (or radius) is called the value or the cost of the solution. Let OPT denote the optimal cost and $\mathcal{C}$ be a corresponding set of k centers. Throughout the paper, we will slightly abuse the notation by using $C \in \mathcal{C}$ not only for the center but also for the corresponding *cluster*. The cluster of a center is a subset of V consisting of the center as well as all the points that have this center as their closest center.

We aim to select centers such that any point is close to at least one center. This can be seen as compressing n data points into k points.

Approximation Algorithms: The k-center problem is NP-hard (Gonzalez 1985), hence the quest for approximation algorithms. A set $\mathcal{S} \subseteq V$ of size k is an α-approximate solution if the maximum distance of any point in V to a point in $\mathcal{S}$ is at most α OPT. Gonzalez (1985) shows that a simple $O(kn)$-time greedy algorithm produces a 2-approximate solution, which is best possible unless P $=$ NP. The running time of this algorithm, however, might be prohibitively slow.

Scalable Distributed Algorithms: Recent work has focused on developing distributed approximation algorithms, in particular in the Massively Parallel Computation (MPC) model. Although inspired by MapReduce (Dean and Ghemawat 2008), this model is also relevant for other distributed computation frameworks, e.g., Spark, Hadoop, Pregel, and Giraph. The aim 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 factor for the quality of the algorithm.

In the MPC model, initially, Ene, Im, and Moseley (2011) presented a constant-round 10-approximation algorithm for k-center with $\Omega(\sqrt{n^{\varepsilon}k^2})$ space per machine. Later, Malkomes et al. (2015) gave a two-round 4-approximation MPC algorithm with $O(\sqrt{nk})$ space per machine and $O(nk)$ total work. Very recently in a surprising work, Ceccarello, Pietracaprina, and Pucci (2018) improved this result to a two-round $(2+\varepsilon)$-approximation MPC algorithm for the k-center problem using $O\big(\sqrt{nk}\big(\frac{4}{\varepsilon}\big)^d\big)$ space per machine and $O(nk)$ total work, where d indicates the dimension of the Euclidean space. Indeed, this result is useful for low-dimensional spaces.

All previous results in this area have two major shortcomings: (i) a total work of $\Omega(nk)$ or $\Omega(n^\epsilon k^2)$, and (ii) consuming $\Omega(k)$ space per machine, which is a byproduct of computing a core-set of size k on each machine. Such algorithms are less practical for large n and k; e.g., to compress $n = 10^9$ data points into $k = 10^6$ data points[1] on 1000 machines, we'd have $nk/1000 = 10^{12}$ operations per machine, which include the costly computation of the distance of two points. All in all, it is impossible to run an $O(nk)$-work algorithm even in a distributed environment, hence the need to develop a new algorithm for that scale.

Moreover, as k grows, we may not have enough space to compute a core-set of size $\Omega(k)$ on a single machine, which is nevertheless the case with all prior work. The importance of algorithms with $o(k)$ space per machine has been observed very recently, in particular, for k-means (Bhaskara and Wijewardena 2018) and k-diversity maximization (Epasto, Mirrokni, and Zadimoghaddam 2019). However, to the best of our knowledge there is no such algorithms for k-center.

In fact, k-center clustering has many applications with a large k. In particular, in the context of semi-supervised learning, when we use label propagation, the number of clusters can be much higher than the number of actual classes. Some examples are spam detection and fraud detection (modeled via binary classification), where many clusters are needed for label propagation to produce a high-quality model. For example, 100M emails labeled by users may easily translate to 1M clusters. Another example in the context of unsupervised learning is same-meaning query clustering for online advertisement or document search (e.g., (Wang et al. 2009)). Indeed, we design distributed algorithms that work on huge datasets, which are orders of magnitude larger than what was discussed in prior *extremal clustering* literature. See the discussion in (Kobren et al. 2017) for more applications.

A natural technique to cope with large instances in distributed environments is extracting a small subset of the input, called *core-set*, which captures the essence of the solution—in the sense that any approximate solution in the subset is also a good solution for the original instance—and applying standard sequential algorithm on this smaller subset, where it is much more efficient. This approach has led to the results mentioned above. In particular, these core-sets use $\Omega(k)$ data points per machine, leading to quadratic total work and large local memory requirement.

1.1 Our Approach and Contribution

In this work, we aim to address the above issues by designing the first highly scalable algorithm for the k-center problem with a sublogarithmic number of rounds of computation, sublinear space per machine, and small total work. Following the lead of (Ceccarello, Pietracaprina, and Pucci 2018), we focus on Euclidean space with constant dimensions.

Theorem 1.1. *There is an $O(\log\log n)$-round MPC algorithm with $\widetilde{O}(n^\varepsilon)$ space per machine and $\widetilde{O}(n^{1+\varepsilon})$ total work that w.h.p.[2] computes an $O(\log\log\log n)$-approximate solution to $O(1)$-dimensional Euclidean k-center with $k(1+o(1))$ centers.*

This result significantly improves the total work and the memory requirement per machine for large k, while it bounds the approximation ratio by $O(\log\log\log n)$ with minor increase in the number of centers. Notice that $\log_2\log_2\log_2 10^{77} \simeq 3$ and $\log_2\log_2\log_2 10^{19725} \simeq 4$. Our algorithm is designed for very large scale where total work $O(nk)$ is not feasible. As expected, the scalability comes at the cost of losing slightly on solution quality.

The main ingredient of our algorithm is the development of significantly smaller core-sets of truly sublinear size. Moreover, instead of purely random partitioning, we introduce a proximity-based partitioning for this problem. Section 5 shows that the traditional core-set method, which distribute the data arbitrarily or uniformly at random, cannot achieve any approximation with core-sets of size less than k, which makes more sophisticated partitioning approaches like ours inevitable.

Finally, we provide an empirical study to corroborate our theoretical guarantees, and demonstrate that the algorithm performs well in practice. In particular, we show that compared to the baselines, (1) our algorithms obtain up to more than 100x speed-up, (2) the solution degrades by at most a factor 2 (as opposed to the $O(\log\log n)$ and $O(\log\log\log n)$ bounds). This is provided in Section 6

1.2 Roadmap of the Paper

Theorem 1.1 is proved in Section 4 by presenting the algorithm DISTRIBUTED-k-CENTER. As a warm-up, we introduce the simplified algorithm UNIFORM-k-CENTER in Section 3. Both algorithms rely on iteratively calling a sampling-based subroutine SAMPLE-AND-SOLVE which we will outline in Section 2. This procedure basically samples a set of *hubs* from the current points and sends each point to its closest hub to form *bags* of points. Then, inside each bag, it invokes the sequential greedy algorithm (i.e., the farthest-point heuristic) for the k-center problem to select centers (which we call *centroids* to distinguish them from the centers in the optimum). The output is the union of these centroids from all bags.

Repeated application of this procedure accumulates some "error" in the final radius.

To adapt to the varying diameter of bags, we synchronize the heuristic runs on their target radius. Notice that the radius of the optimal solution can be guessed at the cost of running $O(\log n)$ parallel algorithms (one of which has the correct estimate of the optimal radius).

The main difference between the simplified algorithm UNIFORM-k-CENTER and our final algorithm DISTRIBUTED-k-CENTER is that the latter changes this estimate of radius in the middle. Intuitively, then, the algorithm first groups absolutely nearby points together ("compressing" each optimal cluster into $O(\log n)$ points) and then solves the k-center problem globally.

[1] For the same-meaning query-clustering application or query-intent modeling, k is often much larger (corresponding to small average cluster size), which makes the problem more acute.

[2] *With high probability*, i.e., with probability at least $1 - \frac{1}{n^c}$, for a sufficiently large constant c.

1.3 Other Related Works

Introduced and studied by (Agarwal, Har-Peled, and Varadarajan 2004), core-sets have been extensively used in designing various distributed algorithms such as k-center (Malkomes et al. 2015; Ceccarello, Pietracaprina, and Pucci 2018), balanced partitioning (Bateni et al. 2014), submodular maximization (Barbosa et al. 2015; Mirrokni and Zadimoghaddam 2015), graph matching (Assadi et al. 2019), k-means and related problems (Bachem, Lucic, and Krause 2015; Lucic, Bachem, and Krause 2016; Bachem et al. 2016; Bachem, Lucic, and Krause 2018; Bhaskara and Wijewardena 2018), and many others (Karloff, Suri, and Vassilvitskii 2010; Lattanzi et al. 2011; Balcan, Ehrlich, and Liang 2013; Indyk et al. 2014). Core-sets have also played a major role in the design of streaming algorithms (Agarwal, Har-Peled, and Varadarajan 2004; Guha et al. 2003).

In the disitributed setting, k-means and k-median (Balcan, Ehrlich, and Liang 2013; Bahmani et al. 2012) as well as balanced clustering has been previously studied (Ugander and Backstrom 2013; Rahimian et al. 2013; Bateni et al. 2014). In particular, Bateni et al. (2014) used a generalization of composable core-sets to study balanced k-center.

Metric k-clustering problems have also been studied in the streaming setting. Charikar et al. (2004) were the first to study k-center in the streaming setting with both insertion and deletion and gave a 8-approximation algorithm using $O(k)$ space. Guha et al. (2003) presented the first single-pass constant-approximation algorithm for k-median in the streaming setting. Despite extensive study of k-means in distributed and streaming settings (Bachem, Lucic, and Krause 2015; Lucic, Bachem, and Krause 2016; Bachem et al. 2016; Bachem, Lucic, and Krause 2018; Bhaskara and Wijewardena 2018), these algorithms are not applicable to the k-center problem.

2 A Proximity-based Sampling Procedure

In this section, we present and analyze a simple sampling-based procedure called SAMPLE-AND-SOLVE, which will serve as building block for our k-center algorithms later on.

The SAMPLE-AND-SOLVE procedure has two stages.

The **bagging stage** (lines 1–2 of Algorithm 1) partitions the points V into *bags* B_h for *hubs* $h \in H$. We sample each point from V independently with probability p, resulting in the set of hubs $H \subseteq V$. Then, we assign each $v \in V$ to its closest hub, breaking ties arbitrarily. A hub h together with the point assigned to it form a *bag* B_h. We place all the points from one bag B_h on the same machine.[3]

The **solving stage** (lines 3–9 of Algorithm 1) marks a set of points in each bag as *centroids* of clusters. We perform the sequential greedy algorithm for k-center (i.e., the farthest-point heuristic) on each bag separately. More precisely, we first mark the bag's original hub as a centroid. Then, iteratively, we mark a non-centroid from the bag with largest distance to the current centroids in the bag, provided that the distance exceeds $2r$.

The bagging stage requires that we efficiently search for nearest neighbors, which is done by a classical application of locality-sensitive hashing (LSH). Using LHS in parallel adds a $\log(\Delta)$ factor to the total work, where Δ is the ratio of the minimum distance to the maximum distance. Due to the space limit we defer the further details to the full version.

[3]With limited number of machines, one may assign the task of several bags to one machine.

Algorithm 1 SAMPLE-AND-SOLVE(V, p, r)

Input: points V, probability $0 < p < 1$, radius r

1: Form H from an i.i.d. sample of V with probability p
2: Assign points in V to their closest hubs from H to form bags $\{B_h : h \in H\}$
3: Process each bag in memory (one per machine)
4: **for** each hub $h \in H$, bag B_h of points assigned to it **do**
5: $\quad S_h \leftarrow \{h\}$
6: $\quad$ **while** $\max_{v \in B_h} \text{dist}(v, S_h) > 2r$ **do**
7: $\quad\quad S_h \leftarrow S_h \cup \{\arg\max_{v \in B_h} \text{dist}(v, S_h)\}$
8: $\quad$ **end while**
9: **end for**

Output: set of centroids $S = \bigcup_h S_h$

2.1 Analysis of Approximation Ratio

We first show that if we choose $r = \text{OPT}$ and have b hubs, we get a 2-approximation with bk centroids. Bear in mind that $p = O(b/n)$ is used in the algorithm. For the sake of simplicity, the write-up assumes the knowledge of OPT. As alluded to before, guessing its correct value only adds a logarithmic factor in space usage and total work.

Lemma 2.1. SAMPLE-AND-SOLVE *produces a solution of cost* $2r$ *with* $|S| \leq bk$ *centroids, when working on* b *bags.*

The number bk of potential centroids, however, might be as large as $\Omega(n)$. Thus, despite the good approximation, the set S does not serve as an acceptable core-set due to its large size. This issue will be resolved by the algorithms UNIFORM-k-CENTER and DISTRIBUTED-k-CENTER in Algorithms 3 and 4, respectively, which iteratively invoke this procedure SAMPLE-AND-SOLVE.

2.2 Analysis of Local Memory Usage

To demonstrate that the local memory usage of SAMPLE-AND-SOLVE is $O\big(\frac{\log n}{p}\big)$ w.h.p., we show that in every iteration w.h.p. the size of each bag is $O\big(\frac{\log n}{p}\big)$. The proof of this lemma contains some novel geometric arguments that simplified the proof. However, due to the space limit we defer the proof to the full version.

Lemma 2.2. *With high probability,* SAMPLE-AND-SOLVE *uses* $O\big(\frac{\log n}{p}\big)$ *space per bag.*

2.3 Number of Centroids in an Optimal Cluster

We conclude with an observation that relates the events of being sampled as hub and being selected as centroid in the SAMPLE-AND-SOLVE with $r = \text{OPT}$. This observation will play a crucial role later in the analysis of UNIFORM-k-CENTER in Section 3. Recall that $\mathcal{C}$ denotes the optimal clustering.

Lemma 2.3. *The number of centroids in each optimal cluster* $C \in \mathcal{C}$ *does not exceed the number of bags, i.e.,* $|S \cap C| \leq$

Algorithm 2 UNIFORM-k-CENTER(V, r)

Input: set V of n points, radius r

1: $S_1 \leftarrow$ SAMPLE-AND-SOLVE(V, $\frac{1}{n^\varepsilon}$, r)
2: $s_0 \leftarrow n^{1-\varepsilon}$
3: **for** $t \leftarrow 1$ **to** $\tau = 3\log\log n$ **do**
4: $\quad s_t \leftarrow \sqrt{s_{t-1}}$
5: $\quad S_{t+1} \leftarrow$ SAMPLE-AND-SOLVE(S_t, $\frac{1}{s_t}$, r)
6: **end for**

Output: S_τ

$|H|$. *Moreover, if at least one hub is sampled from C, then no additional centroids are selected from C. In other words, $S \cap C \subseteq H$ if $H \cap C \neq \emptyset$.*

The following lemma follows easily from the construction of the SAMPLE-AND-SOLVE procedure (in particular, Lines 6–8).

Lemma 2.4. *The clustering produced by* SAMPLE-AND-SOLVE *costs at most 2OPT.*

3 A Simplified Algorithm

In this section, we present a simplified algorithm called UNIFORM-k-CENTER, which iteratively applies SAMPLE-AND-SOLVE, each time collapsing all points of a cluster to its centroid. This leads to a significant reduction of the number of centers from $\Omega(nk)$ to $k(1+o(1))$, while only increasing the approximation ratio from 2 to $O(\log\log n)$.

3.1 Algorithm Description

We let $s_0 = n^{1-\varepsilon}$, $p_0 = 1/n^\varepsilon$, as well as $s_i = \sqrt{s_{i-1}}$ and $p_i = 1/s_i$ for all $1 \le i \le \tau = 3\log\log n$. Starting with $S_{-1} = V$, in iteration $i \ge 0$, we apply SAMPLE-AND-SOLVE on S_{i-1} with radius r and sampling probability p_i to obtain a set of centroids S_i, which will be used as the input set of points for the next iteration. We refer to Algorithm 2 for the pseudocode.

3.2 Analysis of Approximation Ratio

We first note that the centroids selected by this algorithm (and the implicitly induced clustering) indeed approximate the distance of the optimum k-center solution by a factor $O(\log\log n)$. This is a direct consequence of the following lemma, which shows that in every iteration we lose at most an additive 2OPT in the distance, implying that after $\tau = 3\log\log n$ iterations, we have a $6\log\log n$-approximation.

Lemma 3.1. *Any point in V has distance at most $2i$OPT to S_i for each $0 \le i \le \tau$.*

Proof. We prove the claim by induction on i. The statement holds trivially for $i = 0$ where $S_0 = V$. Now suppose, for some $0 \le i \le \tau$ that that any point $v \in V$ has distance at most $2i$OPT to S_i. Take a specific vertex $v \in V$ and let $u \in S_i$ be a point at distance at most $2i$OPT from v. If $u \in S_{i+1}$, then v has distance at most $2i\text{OPT} \le 2(i+1)\text{OPT}$ to S_{i+1}. Consider the case $u \notin S_{i+1}$. Lemma 2.4 guarantees that u is at distance no more than 2OPT from S_{i+1}. □

3.3 Analysis of Number of Centers

We show that we select $k[1+o(1)]$ points as centers.

Lemma 3.2. *The number of selected centers S_τ w.h.p. is $k + O(\log^3 n \log^4\log n)$.*

Observe that if $k = \operatorname{poly}\log(n)$, we can run one of the previous algorithms since $O(k)$ centers would fit in memory and $O(nk)$ total work would be small. So when k is large—the focus of this work—the bound on the number of centers in Lemma 3.2 is indeed $k[1+o(1)]$.

The proof of this lemma is split into two parts corresponding to two phases of the algorithm (which are only relevant for the analysis).

Phase 1: After the first $O(\log\log n)$ iterations of the algorithm, the "compressed" size of each optimum cluster will w.h.p. shrink to $\widetilde{O}(\log n)$. The following lemma is proved in Section 3.3.

Lemma 3.3. *For each optimal cluster $C \in \mathcal{C}$, the remaining number of centroids in C after $O(\log\log n)$ iterations is w.h.p. $\widetilde{O}(\log n)$.*

Phase 2: After each optimal cluster w.h.p. has shrunk to $\widetilde{O}(\log n)$, i.e., after the $O(\log\log n)$ iterations of Phase 1, $O(\log\log n)$ additional iterations suffices to drop the total number of selected centroids to $k[1+o(1)]$. We prove the following lemma in Section 3.3.

Lemma 3.4. *After $O(\log\log n)$ additional iterations, w.h.p., we have only $k + O(k/\log n + \log^2 n \log^2\log n)$ centroids.*

Phase 1

Proof of Lemma 3.3. We show inductively for $i \ge 1$ that w.h.p. the remaining number $|C \cap S_i|$ of centroids in any optimal cluser $C \in \mathcal{C}$ before round i is at most $f_{i-1} = (1+\delta)^{i-1} s_{i-1} \log n \log^2\log n$, for a sufficiently small $\delta = \Theta(1/\log\log n)$.

Before proving the above, let us argue it proves the lemma. Notice that the first term in f_i is $(1+\delta)^i$ which will be $(1+\delta)^{O(\log\log n)} = \exp(O(\delta\log\log n)) = \exp(O(1)) = O(1)$. The second term is $s_i = (n^{1-\epsilon})^{1/2^i}$ which will be $O(n^{1/2^{O(\log\log n)}}) = O(n^{1/O(\log n)}) = O(1)$. Therefore, the size upper bound will be $O(\log n \log^2\log n) = \widetilde{O}(\log n)$, as desired.

For the base case of the induction where $i = 1$, observe that $|C \cap S_1|$ is at most the number of hubs by Lemma 2.3. The expected number of hubs is $s_0 = n^{1-\epsilon}$ since the sampling probability is $1/n^\epsilon$. The Chernoff bound shows that the probability that we sample more than $s_0 \log n$ hubs is at most $e^{-\frac{\log n - 1}{3} n^{1-\varepsilon}} = n^{-\Omega(1)}$. Thus, w.h.p. $|C \cap S_1| \le s_0 \log n$, which proves the base case.

In the inductive step $i > 0$, we suppose $|C \cap S_i| \le f_{i-1}$ and prove that $|C \cap S_{i+1}| \le f_i$ w.h.p. Notice that number of centroids in an optimum cluster is decreasing across rounds, i.e., $C \cap S_{i+1} \subseteq C \cap S_i$, hence $|C \cap S_i| \le f_i$ is trivial. Suppose $|C \cap S_i| > f_i$.

Next we first argue that at least one hub is sampled from each optimal cluster, and then we bound the number hubs sampled from each. Using Lemma 2.3 we then conclude

that no additional centroid is added to each optimum cluster, which finishes the inductive proof.

Hubs are sampled from S_i with probability $p_i = 1/s_i$ at iteration i. Thus the probability that no hub is selected from optimum cluster C is $(1-p_i)^{|C\cap S_i|} \le \exp(-p_i|C\cap S_i|)$. We already have a lower bound on the current compressed size of C, i.e., $|C \cap S_i| > f_i \ge s_i \log n \log^2 \log n$. So the above probability is at most $\exp(-\log n \log^2 \log n) = 1/n^{\Omega(1)}$, that is, w.h.p. one hub is sampled from C.

On the other hand, the expected number of sampled hubs from cluster C at iteration i is

$$\begin{aligned} p_i|C \cap S_i| \le p_i f_{i-1} &= p_i(1+\delta)^{i-1} s_{i-1} \log n \log^2 \log n \\ &= \frac{1}{s_i}(1+\delta)^{i-1} s_{i-1} \log n \log^2 \log n \\ &= \frac{1}{s_i}(1+\delta)^{i-1} s_i^2 \log n \log^2 \log n \\ &= (1+\delta)^{i-1} s_i \log n \log^2 \log n \\ &= f_i/(1+\delta) \qquad (1) \\ &< \log n \log^2 \log n. \qquad (2) \end{aligned}$$

We apply the Chernoff bound, along with (1) and (2), to bound the probability that the number of sampled hubs from cluster C is above f_i as

$$\begin{aligned} \exp(-\frac{\delta^2}{3} f_i/(1+\delta)) &< \exp(-\frac{\delta^2}{3} \log n \log^2 \log n) \\ &= n^{-\frac{\delta^2}{3} \log^2 \log n}, \end{aligned}$$

which is $n^{-\Omega(1)}$ for sufficiently small $\delta = 1/\Theta(\log \log n)$.

As promised, Lemma 2.3 shows $|C \cap S_{i+1}| \le f_i$ w.h.p. Taking union bound over all these small failure probabilities, we conclude the Lemma. □

Phase 2

Proof of Lemma 3.4. At the beginning of Phase 2, each optimal cluster $C \in \mathcal{C}$ has compressed size at most $O(\log n \log^2 \log n)$, by Lemma 3.3. Hence, overall, we have at most $O(k\ell \log n)$ centroids for $\ell = \log^2 \log n$.

Consider an iteration i in Phase 2. Let $y_C = |C \cap S_i|$ be the compressed size of optimal cluster C at the beginning of iteration i. Define $\mathcal{C}_L = \{C \in \mathcal{C} \colon y_C \ge 2\}$ as the set of *large* clusters of compressed size at least 2. Let random variable Z_C denote the compressed size of cluster C after iteration i, and let random variable X_C denote the difference $y_C - Z_C$, i.e., how much the cluster shrinks. Define $y = \sum_{C\in\mathcal{C}_L} y_C$ and $X = \sum_{C\in\mathcal{C}_L} X_C$. The number of centroids in large clusters drops from y to $y - X$ during iteration i.

We will show that w.h.p. $X \ge \gamma y$, for a constant $\gamma > 0$, provided that $y = \Omega(\ell \log^2 n)$ for a sufficiently large hidden constant in Ω. Within $O(\log \log n)$ iterations, y drops from $O(k\ell \log n)$ to $O(\max\{O(k/\log n, \ell \log^2 n\})$. With at most k centroids (by definition) in non-large clusters, we will have a total of $k + O(k/\log n) + O(\ell \log^2 n)$ centroids as a result, proving the lemma.

Recall that $p_i = n^{-(1-\epsilon)/2^i}$ in iteration i. As Phase 2 starts at a sufficiently large round $\Theta(\log \log n)$, we assume $p_i > n^{-1/\Theta(\log n)} > \frac{2}{3}$. On the other hand, since Phase 2 takes another $\Theta(\log \log n)$ iterations, we can upper-bound p_i by some constant $\lambda < 1$.

Take an arbitrary large cluster $C \in \mathcal{C}_L$. We first argue that the success probability of cluster C is $\Pr[1 \le Z_C < \alpha y_C] \ge \beta$ for constants $\alpha < 1$ and $\beta > 0$. (In doing so, we try not to optimize the parameters to keep the argument simple.) Below we bound the number of sampled hubs in C instead of Z_C, since Lemma 2.3 shows that if the former is positive, it serves as an upper bound for the latter.

Let's consider two cases. If $y_C \le 36/(1-\lambda)^2$, only look at the outcome where among all the vertices in C, a particular vertex of C is exclusively sampled as a hub. The success probability is then at least $p_i(1-p_i)^{y_C - 1} \ge \beta_0 = \frac{2}{3}(1-\lambda)^{36/(1-\lambda)^2}$ which is a constant (albeit small). Let $\alpha_0 = \frac{1}{2}$ to have $\Pr[1 \le Z_C < \alpha_0 y_C] \ge \beta_0$. In the second case, $y_C > 36/(1-\lambda)^2$. The probability that no hub from C is sampled in this case is at most $1 - p_i \le \frac{1}{3}$. When at least one hub is sampled, the expected number of sampled hubs is $\mathbb{E}[Z_C | Z_C > 0] = p_i y_C \le \lambda y_C$. We apply the Chernoff bound to get

$$\begin{aligned} \Pr\left[Z_C > \frac{\lambda+1}{2} y_C \middle| Z_C > 0\right] &\le \exp(-\frac{(1-\lambda)^2}{12} p_i y_C) \\ &\le \exp(-\frac{(1-\lambda)^2}{12}\frac{2}{3} y_C) \\ &\le \exp(-\frac{(1-\lambda)^2}{18} y_C) \\ &\le \exp(-2). \end{aligned}$$

The success probability in this case is at least $1 - \frac{1}{3} - e^{-2} > 0.53$. Thus, in this case, we have $\Pr[1 \le Z_C \le \alpha_1 y_C] > \beta_1 = 0.53$ for $\alpha_1 = (\lambda+1)/2$. Therefore, for $\alpha = \max(\alpha_0, \alpha_1)$ and $\beta = \min(\beta_0, \beta_1)$, we have

$$\Pr[1 \le Z_C \le \alpha y_C] > \beta. \qquad (3)$$

Since $C \cap S_i$ is decreasing through the algorithm, a non-large cluster (of compressed size one) may not become large again. Clearly $X_C \ge 0$ for any cluster C. We showed in (3) that $X_C \ge (1-\alpha) y_C$ with probability β for any large cluster C. Taking expectations and summing over large clusters yields $\mathbb{E}[X] \ge \beta(1-\alpha) y \ge \beta(1-\alpha)\Theta(\ell \log^2 n)$, where the last derivation uses the assumption $y = \Omega(\ell \log^2 n)$ when we need to guarantee progress. Since $0 \le X_C \le y_C$ and $y_C = O(\ell \log n)$ by Lemma 3.3, the Hoeffding inequality gives

$$\begin{aligned} \Pr\left[X \le \frac{1}{2}\mathbb{E}[X]\right] &\le \exp\left(-\frac{2|\mathcal{C}_L|\,\mathbb{E}^2[X]}{4\big(\max_C\{y_C\}\big)^2}\right) \\ &\le \exp\left(-\frac{\Theta(\ell^2 \log^4 n)}{2O(\ell^2 \log^2 n)}\right) \\ &\le \exp(-\Omega(\log^2 n)) = n^{-\Omega(\log n)}. \end{aligned}$$

Therefore, provided that $y = \Omega(\ell \log^2 n)$ for a sufficiently large hidden constant, Phase 2 reduces, w.h.p., the total number of centroids in large clusters by a constant factor, i.e., $X \ge \gamma y$ where $\gamma = \beta(1-\alpha)/2$. This concludes the proof of the lemma. □

Algorithm 3 DISTRIBUTED-k-CENTER(V, r)

Input: set V of n points, radius r

Phase 1:
$S_1 \leftarrow$ UNIFORM-k-CENTER(V, $\frac{r}{\log\log n}$)
Phase 2:
$s_1 \leftarrow \log n \log\log n$
for $t \leftarrow 1$ **to** $\tau = c_t \log\log\log n$ **do**
  $S_{t+1} \leftarrow$ SAMPLE-AND-SOLVE(S_t, $\frac{1}{s_t}$, r)
  $s_{t+1} \leftarrow s_t^{2/3}$
end for

Output: S_τ

4 The Final Algorithm

In this section, we prove Theorem 1.1 by showing that DISTRIBUTED-k-CENTER, is an $O(\log\log\log n)$-approximation algorithm. This algorithm differs from UNIFORM-k-CENTER of Section 3, in that it uses a non-uniform estimate of the optimum throughout the algorithm.

4.1 Algorithm Description

The algorithm consists of two phases.

Phase 1: In the first phase, we run UNIFORM-k-CENTER from Section 3 with $r = \frac{\text{OPT}}{\log\log n}$, to obtain a set S_1 of initial centroids. As we will see, this compresses each optimal cluster to polylogarithmic size without significant effect on the cost.

Phase 2: The second phase iteratively applies the SAMPLE-AND-SOLVE procedure from Section 2 on S_i with $r = \text{OPT}$ and sampling probabilities $1/s_i$ where $s_1 = \log n \log\log n$ and $s_i = s_{i-1}^{2/3}$ for $i > 1$. After $\tau = c_t \log\log\log n$ iterations, for a sufficiently large constant c_t, we output the set of centroids S_τ.

The algorithm is analyzed in two steps, corresponding to the phases of the algorithm.

4.2 Analysis of Phase 1

The first phase has the purpose of reducing the number of centroids in each optimal cluster $C \in \mathcal{C}$ to $\widetilde{O}(\log n)$, while preserving the approximation.

Lemma 4.1. *W.h.p. the set S_1 of centroids induces a constant-approximate clustering with the property that $C \cap S_1 = O(\log^2 n \log^2\log n)$ for each $C \in \mathcal{C}$.*

4.3 Analysis of Phase 2

The second phase reduces the number of centroids to $k[1 + o(1)]$ in only $O(\log\log\log n)$ rounds, as opposed to $O(\log\log n)$ rounds of UNIFORM-k-CENTER, which then yields an overall approximation ratio of $O(\log\log\log n)$, as opposed to $O(\log\log n)$.

Approximation Ratio: It follows from Lemma 3.1 that the final clustering S_τ is a $2\tau = O(\log\log\log n)$-approximation to the implicit solution in S_1, as we lose an additive factor in every iteration. Since, by Lemma 4.1, S_1 is an $O(1)$-approximation to the optimum, the output of the second phase is an $O(\log\log\log n)$-approximation.

k	T	C	L	S_C	S_L
5,000	7	34	8	4.9	1.1
10,000	7	63	13	8.5	1.7
20,000	8	123	31	15.6	3.9
50,000	8	304	130	36.2	15.4
100,000	9	614	393	67.5	43.2
200,000	11	1,267	1,206	120.0	114.1

k	T	C	S_C
1,000	63	380	6.0
5,000	129	698	5.4
10,000	289	1,096	3.8
50,000	321	4,282	13.4
100,000	322	8,585*	26.6
500,000	333	40,807*	122.6

Figure 1: Speed-up for `en-wiki` (top) and `prod` (bottom). All times are rounded to the closest minute. The running times of DISTRIBUTED-k-CENTER, classic greedy and lazy greedy are denoted by T, C, L, respectively. The runtimes marked with asterisks are extrapolated, as they would take one or more weeks to finish. Then S_C and S_L show the speed-up that DISTRIBUTED-k-CENTER achieves over the particular greedy algorithm.

Number of Selected Centers: In order to prove Theorem 1.1, it remains to show $|S_\tau| \leq k[1 + o(1)]$. This is presented in the following lemma. The proof of this lemma is partly similar to that of Lemma 3.4.

Lemma 4.2. *After $O(\log\log n)$ additional iterations, w.h.p., we have only $(1 + o(1))k$ centroids.*

5 Lower Bounds

In this section we state our lower bounds. Due to space limit we defer the proofs to the full version. Next lemma states that random partitioning leads to bad core-sets, hence our more sophisticated partitioning method is necessary.

Lemma 5.1. *With a random partition of points into $m < k$ bags, the core-set needs at least k points in each bag.*

Next lemma states that our more sophisticated partitioning technique does not immediately lead to smaller core-sets.

Lemma 5.2. SAMPLE-AND-SOLVE *needs at least k centroids from each bag—at least bk overall—to guarantee a 2-approximation.*

6 Empirical Study

Dataset	# points	dimension	norm
`song`	515,345	90	variable
`en-wiki`	3,831,716	100	1.0
`prod`	108,729,118	64	1.0

Table 1: Our datasets at a glance.

We run the proposed algorithms on public and private datasets in order to demonstrate that (1) the new algorithms are pretty scalable, and (2) they produce high-quality solutions.

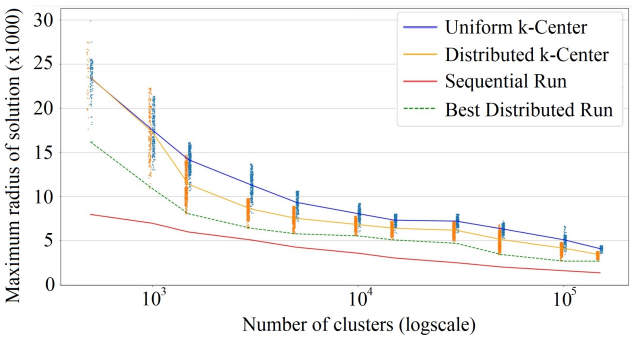

Figure 2: Comparing quality of our algorithms to the baseline sequential greedy algorithm on `song` dataset. The red curve (at the bottom) shows the maximum radius of the solution produced by the baseline for 11 values of k (i.e., number of clusters) ranging from 500 to 150K. Right above the red curve, the green curve plots the best solution we obtained from either of our algorithms. The blue and orange dots represent various runs of the two algorithms, with the curves of same color fitting to the mean quality at each value of k.

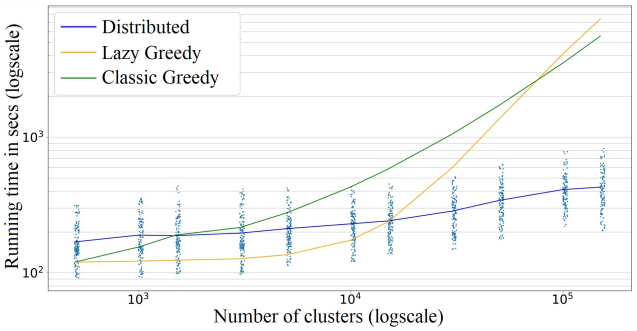

Figure 3: Comparing the running time of our algorithms to the baseline sequential greedy algorithm on `song` dataset. The blue dots represent the running time (in seconds) of our algorithm. The blue curve denotes the mean for different values of k (i.e., number of clusters). The orange and green curves show the running time of two implementations of the sequential greedy algorithm (i.e., "lazy" and "classic").

Datasets. We employ 3 datasets in the experiments: two publicly available datasets (`song` (Dheeru and Karra Taniskidou 2017) and `en-wiki` (Epasto, Mirrokni, and Zadimoghaddam 2017)) and a much larger private one (`prod`). We provide further details of the datasets in the full version.

Solution quality. The approximation factors, $O(\log \log n)$ and $O(\log \log \log n)$, which we established for UNIFORM-k-CENTER and DISTRIBUTED-k-CENTER, are small constants for any imaginable dataset. For instance, both are less than 6 for a graph with 10^{18} vertices. The experiments show that the hidden constants are not big either, as we can always get within a factor 3 of the output of the sequential greedy algorithm. As expected, DISTRIBUTED-k-CENTER comes up a little ahead in terms of solution quality, at the cost of a slight increase in running time.

We implemented the two algorithms in C++, and ran each multiple times for every value of k, on a cloud platform (similar to Hadoop) which implements the MapReduce framework. See Figure 2 for the results. The ratio of the quality of the best solution we obtain over that of the greedy algorithm ranges from 1.2 to 2.0 (corroborating the theoretical results and bounding the hidden constants). The difference in quality among results for each value of k is due to a couple of factors: (1) We try a few ways to set the parameters (e.g., the precise sampling factor), which adds some diversity to the results. (2) Although the theoretical analysis of our algorithm (in particular, the LSH part) works for small dimensions, we try the algorithms for real datasets with high dimensions. Despite the resulting randomness, the experiments demonstrate the desired concentration bounds hold. In fact, computing the optimal radius given a fixed set of center points is not easy to do precisely faster than the naive $O(kn)$ brute-force method. Therefore, in contrast to the sequential algorithm, the quality of our solutions might be better than reported.

Scalability. We compare the running time of our implementation to that of the sequential greedy algorithm. The reported times for the sequential algorithm are from C++ implementations that ran on a dedicated computer without any other computationally intensive task at the time: *Intel(R) Xeon(R) W-2135 CPU @ 3.70GHz with 6 cores and 8MB cache*. The reported times for distributed algorithms correspond to running on a cloud platform, using no more than 100 machines, each of which has a weaker CPU than the machine used by the sequential algorithm. However, our code did run in a silo, and in fact, it competed with many other tasks using the same cloud infrastructure, so it suffered from scheduling delays (for each MPC round) as well as preemptions. Moreover, no attempt was made to optimize this code. Therefore, the comparison is not *apples to apples*, and it overestimates the runtime of the distributed algorithm. As noticed in Figure 3, the running time for a particular k typically has a multiplicative range of 3. The actual running time of the distributed algorithm would be even smaller than the fastest sample point, if there were no competing tasks and we used the 100 machines at all times.

In light of the above, we still see that our distributed algorithms outperform the sequential algorithm, specially when k is large. For $k = 1.5 \times 10^5$, our algorithm can run in less than 200 seconds, while both greedy algorithms take more than 5000 seconds—a 25x speed-up.

Figure 1 presents the speed-up numbers for `en-wiki` and `prod`. For the `prod` dataset, we only ran the sequential greedy algorithm for $k \leq 75,000$, in which case it took more than 100 hours. We extrapolated[4] to obtain the latter two entries in the corresponding table, as it would take a long time. Even with the scheduling delays and shuffle times, the distributed algorithm produces significant speed-up over the sequential method. The speed-up ranges from about 5x to over 100x.

[4]This estimate is based on the actual runtimes for the `song` dataset and the actual runtimes for the `prod` dataset (for k up to 75,000) as well as the amount of data in `prod`, and the $O(kn)$ time complexity of the greedy algorithm.

References

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

Assadi, S.; Bateni, M.; Bernstein, A.; Mirrokni, V.; and Stein, C. 2019. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1616–1635. SIAM.

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

Bachem, O.; Lucic, M.; and Krause, A. 2015. 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*, 209–217. URL http://jmlr.org/proceedings/papers/v37/bachem15.html.

Bachem, O.; Lucic, M.; and Krause, A. 2018. 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*, 1119–1127. doi:10.1145/3219819.3219973. URL https://doi.org/10.1145/3219819.3219973.

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

Balcan, M.-F. F.; Ehrlich, S.; and Liang, Y. 2013. Distributed k-means and k-median Clustering on General Topologies. In *Advances in Neural Information Processing Systems*, 1995–2003.

Barbosa, R.; Ene, A.; Nguyen, H.; and Ward, J. 2015. The power of randomization: Distributed submodular maximization on massive datasets. In *International Conference on Machine Learning*, 1236–1244.

Bateni, M.; Bhaskara, A.; Lattanzi, S.; and Mirrokni, V. 2014. Distributed balanced clustering via mapping coresets. In *Advances in Neural Information Processing Systems*, 2591–2599.

Bhaskara, A.; and Wijewardena, M. 2018. Distributed Clustering via LSH Based Data Partitioning. In *International Conference on Machine Learning*, 569–578.

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

Charikar, M.; Chekuri, C.; Feder, T.; and Motwani, R. 2004. Incremental clustering and dynamic information retrieval. *SIAM Journal on Computing* 33(6): 1417–1440.

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

Dheeru, D.; and Karra Taniskidou, E. 2017. UCI Machine Learning Repository. URL http://archive.ics.uci.edu/ml, Accessed on 01/07/2019.

Ene, A.; Im, S.; and Moseley, B. 2011. Fast clustering using MapReduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 681–689. ACM.

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

Epasto, A.; Mirrokni, V. S.; and Zadimoghaddam, M. 2017. Bicriteria Distributed Submodular Maximization in a Few Rounds. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 25–33. doi:10.1145/3087556.3087574. URL https://doi.org/10.1145/3087556.3087574.

Gonzalez, T. F. 1985. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science* 38: 293–306.

Guha, S.; Meyerson, A.; Mishra, N.; Motwani, R.; and O'Callaghan, L. 2003. Clustering data streams: Theory and practice. *IEEE transactions on knowledge and data engineering* 15(3): 515–528.

Indyk, P.; Mahabadi, S.; Mahdian, M.; and Mirrokni, V. S. 2014. Composable core-sets for diversity and coverage maximization. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 100–108. ACM.

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

Kobren, A.; Monath, N.; Krishnamurthy, A.; and McCallum, A. 2017. A hierarchical algorithm for extreme clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 255–264. ACM.

Lattanzi, S.; Moseley, B.; Suri, S.; and Vassilvitskii, S. 2011. Filtering: a method for solving graph problems in mapreduce. In *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*, 85–94. ACM.

Lucic, M.; Bachem, O.; and Krause, A. 2016. 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*, 1–9. URL http://jmlr.org/proceedings/papers/v51/lucic16.html.

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

Mirrokni, V.; and Zadimoghaddam, M. 2015. Randomized composable core-sets for distributed submodular maximiza-

tion. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, 153–162. ACM.

Rahimian, F.; Payberah, A. H.; Girdzijauskas, S.; Jelasity, M.; and Haridi, S. 2013. Ja-be-ja: A distributed algorithm for balanced graph partitioning. In *Self-Adaptive and Self-Organizing Systems (SASO), 2013 IEEE 7th International Conference on*, 51–60. IEEE.

Ugander, J.; and Backstrom, L. 2013. Balanced label propagation for partitioning massive graphs. In *Proceedings of the sixth ACM international conference on Web search and data mining*, 507–516. ACM.

Wang, H.; Liang, Y.; Fu, L.; Xue, G.-R.; and Yu, Y. 2009. Efficient query expansion for advertisement search. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, 51–58. ACM.