Brief Announcement: Fast and Better Distributed MapReduce Algorithms for k-Center Clustering

Sungjin Im
University of California, Merced
Merced, CA 95343
sim3@ucmerced.edu

Benjamin Moseley
Washington University in St. Louis.
St. Louis, MO 63130
bmoseley@wustl.edu

ABSTRACT

We revisit the k-center clustering problem in MapReduce. Previously, a 10-approximation algorithm that runs in $O(1)$ rounds was known. In this work, we present two 2-approximate MapReduce algorithms that run in 3 or 4 rounds. These algorithms are essentially the best one can hope for in terms of both approximation factor and round complexity. We then consider the k-center problem with outliers for the first time in the MapReduce setting. For this problem, we introduce a 4-approximate 3-round MapReduce algorithm.

Categories and Subject Descriptors

F.2.2 [**Nonnumerical Algorithms and Problem**]: Computations on discrete structures

General Terms

Algorithms, Theory

Keywords

MapReduce, Clustering, k-center, Outliers.

1. INTRODUCTION

Clustering is a fundamental problem faced in a variety of fields and, due to this, there is a vast literature on the topic. See [14] for a survey on the literature on clustering. Generally, in a clustering problem, the goal is to partition, or cluster, a given set of points such that the points belonging to the same cluster are similar. Clustering provides a useful summary of large data sets and often serves as a key component in numerous applications arising in data mining, information retrieval, bioinformatics, and social network analysis.

One of the most basic and well-understood clustering problems is the k-center problem. In the k-center problem, the input consists of an integer k, a set U of n points along with pairwise distance $d(u, v)$ between any two points $u, v \in U$. This distance represents the dissimilarity between the points – similar points are close together while dissimilar points are farther away from each other. It is generally assumed that the data points lie in a metric space. The points in U are to be partitioned into k clusters. The partitioning of points into the clusters can be done explicitly, but alternatively can be done by choosing a set of k points. That is, to define the clustering, one chooses a set $C \subseteq U$ of k points from U which are called *centers*. For a subset $S \subseteq U$ let $d(v, S) = \min_{u \in S} d(u, v)$ be the minimum distance of a point $v \in U$ to a point in S. Choosing the set $C = \{c_1, c_2, \ldots c_k\}$ of k centers naturally defines a partitioning of U into k sets $P_1, P_2, \ldots P_k$ by setting P_i to contain each point $v \in U$ where $d(v, C) = d(v, c_i)$. Here ties are broken arbitrarily in the assignment of points to clusters. The goal is to choose the set C of k centers such that $\max_{u \in U} d(u, C)$ is minimized.

The complexity of the metric k-center problem is well-understood. The problem is NP-Hard and further for any $\epsilon > 0$, no $2-\epsilon$-approximation is possible unless P = NP [6,8]. There are 2-approximate algorithms known, giving the best worst case analysis result one could hope for [6, 8]. Besides these results, previous work has considered generalizations of the problem. One of the most popular generalizations is the k-center with *outliers* problem [4]. In this problem, the setting is the same, but additionally an algorithm is allowed to discard up to $z \geq 0$ points from U without penalty. The set of discarded points needs not be included in the partitioning and do not contributed to the objective function. Here z is a parameter input to the problem. In this problem, it is assumed that the data is noisy, which occurs frequently in practice. Unfortunately the quality of clustering solutions can dramatically change based on a small number of outliers data points and, due to this, the k-center with outliers problem has been considered. For this generalization of the k-center problem a 3-approximation algorithm is known [4].

Clustering Large Data: Today, a fundamental application of clustering is for analyzing large data sets. Unfortunately, once data sets become large enough, most sequential algorithms are rendered ineffective. This is due to the time constraints when running a sequential algorithm on the large data sets and also due to the memory constraints; a single machine may not have enough memory to fit the entire data set for a problem. For these reasons, when datasets become large, practitioners turn to alternative computational frameworks for data processing, such as distributed computing.

The MapReduce [13] paradigm has emerged as the de facto standard distributed computing framework for pro-

SPAA'15, June 13–15, 2015, Portland, OR, USA.
ACM 978-1-4503-3588-1/15/06.
http://dx.doi.org/10.1145/2755573.2755607.

cessing large data sets. A MapReduce computation begins with data being randomly (or arbitrarily) assigned to a set of machines. The data is processed in successive rounds. During a round, the dataset is processed in parallel across the machines without inter-machine communication. Communication occurs between the machines only during successive rounds. Along with being widely used in practice, MapReduce has recently been of interest to the theoretical and machine learning communities [1–3, 5, 7, 9–12]. A formal computational model of the MapReduce framework was given in [9]. The main parameters of the model are that the number of machines and memory on the machines are sublinear in the input size to the problem; a natural constraint for problems where the data is assumed to be large. The computation performed during a round is assumed to take a polynomial time and the goal is to minimize the number of rounds, the main time bottleneck in MapReduce computations. See [10] for details of the formal model. Ideally, an algorithm runs in a small constant number of rounds. Notice that by forcing the machine to have sublinear memory renders the adaptation of sequential algorithms to the MapReduce setting challenging. Further, the formal parameters of the model enforce the constraint that no machine ever sees the entire input to the problem over the entire computation, forcing the algorithm designer to use a highly parallelized algorithm.

The MapReduce framework is widely used by practitioners for clustering large data sets [5, 10, 15]. Unfortunately, known sequential algorithm techniques for the k-center problem cannot be efficiently implemented in the MapReduce setting. For example, one well known k-center algorithm starts with a set $C = \emptyset$. The algorithm begins by adding an arbitrary point form U to C. Then, the algorithm iteratively adds the point u from U to C such that $d(u, C)$ is maximized until $|C| = k$. Any naive implementation of the this algorithm in the MapReduce setting would be quite inefficient, running in $\Theta(k)$ rounds. This is because in each step of the algorithm, the point chosen is dependent on the previous points chosen to be in C. Due to this, an adaptation of the algorithm to MapReduce would only add a single point to C in one round.

To have an fast efficient MapReduce algorithm for the k-center problem, it is required that many points can be added to the solution set in one round. However, to do this, new algorithmic techniques are required beyond known sequential techniques. Previously, [5] considered the k-center clustering in MapReduce for the first time. This work showed an $O(1)$-round algorithm by iteratively sampling from the universe U to construct a sublinear sized set S that is a sketch of the original universe U. After constructing S in a distributed fashion, the algorithm then clusters S on a single machine using a known sequential k-center algorithm, resulting in a 10-approximation for the k-center problem in MapReduce in $O(1)$ rounds.

The work of [5] was the first to address the k-center problem in MapReduce theoretically. However, the work left several open questions. In particular, is there an algorithm that returns the best possible 2-approximate clustering in $O(1)$-rounds? Further, the k-center with outliers problem has not been previously considered in MapReduce. Can one design an efficient approximation algorithm for k-center with outliers in MapReduce?

Results: In this work, we consider the k-center problem in the MapReduce setting and its generalization with outliers. Throughout this work, we assume that the distance between points is given by oracle access to the function d and that the oracle can be stored on the machines without using additional memory. This is typically the case, for instance, if the points are in Euclidean space. Our work begins by showing a computationally simple MapReduce algorithm that gives the best possible 2-approximation to the k-center problem in MapReduce if the value of the of the optimal solution, OPT, is known in advance. This algorithm deviates from previous work [5] by not constructing a sketch of the universe U, but rather using a sampling technique to effectively simulate a known sequential greedy algorithm the k-center problem.

For all of our algorithms, we state the minimum memory the algorithm requires on the machines, which is sublinear. All of our algorithms only require that the total number of machines is such that the entire dataset can be stored across all the machines. We note that the memory requirement implicitly assumes that k, the number of clusters, is small. For problems where the number of clusters is pre-specified, it is almost always the case that the number of clusters is assumed to be small [5, 10]. This is because, otherwise, one typically considers a clustering problem where the algorithm determines the clusters, since when the number of clusters is large one usually is not aware of the 'right' number of clusters a priori.

Theorem 1. *There exists a 3-round MapReduce algorithm for k-center that when knowing* OPT *returns a 2-approximate solution. The algorithm uses $O(kn^{1/2} \log n)$ memory on each machine w.h.p. Further, the total communication required is at most $O(kn^{1/2} \log n)$ w.h.p. assuming that the data is initially stored on the machines.*

This result does require knowledge of the value of the optimal solution, OPT. However, the algorithm can guess the value of the optimal solution and run the algorithm for each guess of the optimal solution in parallel. Let Δ be the maximum distance between two points of U and the minimum distance be 1. Notice that value of Δ is an upper bound on the optimal solution's cost. The algorithm can geometrically guess the value of the optimal solution by powers of $(1+\epsilon)$ for some $\epsilon > 0$, $(1+\epsilon)^0, (1+\epsilon)^1, ..., (1+\epsilon)^{\log_{1+\epsilon}\Delta}$, and run the algorithm for each possible guess. By increasing the communication by a $O(\log_{1+\epsilon}\Delta)$ factor and the approximation $(1+\epsilon)$ factor, the algorithm can simulate the knowledge of OPT thereby yielding a $2(1+\epsilon)$-approximate solution.

Unfortunately, this result could be seen as unsatisfactory if the maximum distance between points, Δ is very large since one cannot preform the $O(\log_{1+\epsilon}\Delta)$ guesses of OPT in parallel. Due to this, we extend the previous algorithm to not require information about the value of the optimal solution. Here we show an intricate way to circumvent the actual knowledge of OPT by using an additional round.

Theorem 2. *There exists a 4-round MapReduce algorithm for k-center that returns a 2-approximate solution and uses memory at most $O(kn^{1/2} \log n)$ on each machine w.h.p. Further, the total communication required is at most $O(k^2 n^{1/2} \log n)$ w.h.p. assuming that the data is initially stored on the machines.*

Finally, we consider the k-center problem with outliers for the first time in the MapReduce setting. Here we give a result assuming that the algorithm guesses the value of the optimal solution in parallel. As before, we can simulate the knowledge of OPT by guessing the value of OPT and running the algorithm in parallel for each guess. Here the algorithm requires an additional z factor in memory. Recall that z is the number of outliers. Although it could be the case that the number of outliers is large in practice, our work shows that if z is smaller than $O(n^{1/2-\epsilon})$ then there is an efficient MapReduce algorithm that uses at most $\tilde{O}(n^{1-\epsilon})$ memory for any $\epsilon > 0$.

THEOREM 3. *There exists a 3-round MapReduce algorithm for k-center with outliers that when knowing* OPT *returns a 4-approximate solution and uses memory at most $O(kzn^{1/2}\log n)$ on each machine w.h.p. Further, the total communication required is at most $O(kzn^{1/2}\log n)$ w.h.p. assuming that the data is initially stored on the machines.*

2. K-CENTER KNOWING OPT

In this section, to give a taste of our results, we present an algorithm, along with a sketch of the analysis, that produces a 2-approximate solution for the k-center problem in 2-rounds assuming it knows the value, OPT, of the optimal solution. If the maximum distance between any two points is polynomially bounded then by guessing $\Theta(\log_{1+\epsilon}\Delta)$ values for OPT and running this algorithm for each guess of OPT in parallel the algorithm will produce a $(2+\epsilon)$-approximation in $O(1)$-rounds for any $\epsilon > 0$.

Algorithm: Set $S = S_1 = S_2 = \emptyset$. The algorithm samples each point in U independently with probability $1/n^{1/2}$ and maps them to a single machine. Let X_1 be the set of sampled points. The sampling can be done in the first map phase. In the reduce phase, the machine with the sampled points runs the following greedy algorithm. It starts with a solution set S and first adds an arbitrary point from X_1 to S. Then it iteratively adds any point $i \in X_1$ to S if $d(S,i) > 2\mathrm{OPT}$ and $|S| < k$. Let S_1 be the set S the algorithm has computed when no point in X_1 could be added to S. If $|S_1| = k$, then the algorithm output S_1. Otherwise, in the next round, the algorithm checks for each points i if $d(S_1,i) > 2\mathrm{OPT}$. Let $X_2 := \{i \in U \mid d(S_1,i) > 2\mathrm{OPT}\}$ be this set of points. One can determine X_2 by sending S_1 to each machine and if the condition $d(S_1,i) > 2\mathrm{OPT}$ holds for each point in U on this machine. Then the algorithm maps all the points in X_2 and S_1 to a single machine. Let $S_2 = S_1$. The points in X_2 are iteratively added to S_2 where, at any time, a point $i \in X_2$ is added if $d(i,S_2) > 2\mathrm{OPT}$ and $|S_2| < k$. Let S_2 be the resulting set. The algorithm outputs S_2.

Sketch of Analysis: We first show that X_1 includes $O(n^{1/2}\log n)$ points w.h.p. This is an easy consequence of standard Chernoff bounds. Then, we bound the size of the set X_2: If $|S_1| < k$ after the first phase of the algorithm, then set X_2 includes at most $O(kn^{1/2}\log n)$ points w.h.p. Intuitively, there cannot be many points that are 'far' from the points in X_1 w.h.p. This is because the algorithm then would have sampled some of these points. Then, the memory usage and communication cost claimed in Theorem 1 easily follow.

Now it remains to show our algorithm gives a 2-approximation. Let S^* be the set output by the algorithm, which is output as S_1 at the end of the first phase if $|S_1| = k$ and otherwise $S^* = S_2$ at the end of the algorithm. Notice that whenever the algorithm adds a point to the set S_1 it never gets removed. The same holds for S_2 and $S_1 \cup S_2 = S^*$. Further, notice after the first iteration of the algorithm, any point i such that $d(S_1,i) > 2\mathrm{OPT}$ is in the set X_2. Thus, the only points discarded after the first phase are close to the set S_1. Say that there is some point $i \in U$ such that $d(S^*,i) > 2\mathrm{OPT}$. Since i would have been in the set X_2, it must be the case that $|S^*| = k$. Further, knowing that a point is only added to the solution if its distance is greater than 2OPT from the points already in the solution set, there must be $k+1$ points of distance greater than 2OPT from each other. Two of these points must be served by the same center in the optimal solution, but, by the triangle inequality, this contradicts the definition of OPT.

Acknowledgements. The first author's research was supported in part by NSF grant CCF-1409130.

3. REFERENCES

[1] A. Andoni, A. Nikolov, K. Onak, and G. Yaroslavtsev. Parallel algorithms for geometric graph problems. In *STOC*, pages 574–583, 2014.

[2] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii. Scalable k-means++. *PVLDB*, 5(7):622–633, 2012.

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

[4] M. Charikar, S. Khuller, D. M. Mount, and G. Narasimhan. Algorithms for facility location problems with outliers. In *SODA*, pages 642–651, 2001.

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

[6] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38(0):293 – 306, 1985.

[7] M. T. Goodrich, N. Sitchinava, and Q. Zhang. Sorting, searching, and simulation in the mapreduce framework. In *ISAAC*, pages 374–383, 2011.

[8] D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

[9] H. J. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *SODA*, pages 938–948, 2010.

[10] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani. Fast greedy algorithms in mapreduce and streaming. In *SPAA*, pages 1–10, 2013.

[11] S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii. Filtering: A method for solving graph problems in MapReduce. In *SPAA*, pages 85–94, 2011.

[12] B. Mirzasoleiman, A. Karbasi, R. Sarkar, and A. Krause. Distributed submodular maximization: Identifying representative elements in massive data. In *NIPS*, pages 2049–2057, 2013.

[13] T. White. *Hadoop: The Definitive Guide.* O'Reilly Media, 2009.

[14] R. Xu and D. Wunsch. Survey of Clustering Algorithms. *IEEE Trans Neural Netw*, 16(3):645–678, 2005.

[15] W. Zhao, H. Ma, and Q. He. In M. G. Jaatun, G. Zhao, and C. Rong, editors, *CloudCom.*