Distributed Balanced Clustering via Mapping Coresets

MohammadHossein Bateni Aditya Bhaskara Silvio Lattanzi Vahab Mirrokni

Google Research NYC

Abstract

Large-scale clustering of data points in metric spaces is an important problem in mining big data sets. For many applications, we face explicit or implicit size constraints for each cluster which leads to the problem of clustering under capacity constraints or the "balanced clustering" problem. Although the balanced clustering problem has been widely studied, developing a theoretically sound distributed algorithm remains an open problem. In the present paper we develop a new technique based on "mapping coresets" to tackle this issue. Our technique results in first distributed approximation algorithms for balanced clustering problems for a wide range of clustering objective functions such as k-center, k-median, and k-means.

A core-set for a point-set in a metric space is a subset of the point-set with the property that an approximate solution to the whole point-set can be obtained given the coreset alone. A mapping coreset is coreset with an additional mapping of points in the original space to points in the coreset. In addition to solutions of the coresets, the mapping of points will later be used to reconstruct the solution to the original problem. In this paper, we first show that using mapping coresets, one can turn a single-machine α-approximation algorithms for balanced clustering problem to an efficient distributed $O(\alpha)$-approximation algorithm for the same problem in massive data processing frameworks such as the MapReduce framework and streaming algorithms. Furthermore, we develop a mapping coreset for a general class of balanced clustering problems. Using this technique, we will develop the first constant-factor approximation algorithms for balanced k-center, k-median, and k-means running in a constant number of rounds. Moreover even for some general varaints of the uncapacitated clustering problems our general framework implies the first constant-factor distributed approximation algorithm in a constant number of rounds.

1 Introduction

Large-scale clustering of data points in metric spaces is an important problem in mining big data sets. Many variants of such clustering problems have been studied spanning, for instance, a wide range of ℓ_p objective functions including the k-means, k-median, and k-center problems. Distributed algorithms are the only plausible approach to clustering big data sets where the data does not fit on a single machine, Motivated by a variety of big data applications, distributed clustering has attracted significant attention over the literature [14, 4, 5]. In fact, several practical algorithms have been developed for this problem, some of which have reasonable theoretical approximation guarantees. In many of these applications, though, an explicit or implicit size constraint is imposed for each cluster; e.g., if we cluster the points such that each cluster fits on one machine, the size constraint is enforced by the storage constraint on each machine. We refer to this as *balanced* clustering. In the setting of network location problems, these are referred to as *capacitated* clustering problems [7, 21, 22, 12, 3]. The distributed balanced clustering problem is also very well-studied and several distributed algorithms have been developed for it in the context of large-scale graph partitioning [27, 26][1]. Despite this extensive literature, none of the distributed algorithms developed for the balanced version of the problem have theoretical approximation guarantees. The present work

[1] A main difference between the balanced graph partitioning problems and balanced clustering problems considered here is that in the graph partitioning problems a main objective function is to minimize the cut function.

presents the first such distributed algorithms for a wide range of balanced clustering problems with provable approximation guarantees. To acheive this goal, we develop a new technique based on *mapping coresets*.

A coreset for a set of points in a metric space is a subset of these points with the property that an approximate solution to the whole point-set can be obtained given the coreset alone. Using coresets is a popular approach to processing massive data: one can first extract a compact representation of the data and then perform further processing only on the representation itself. This approach may result in reducing the cost of processing, communicating and storing the data, as the representation size can be much smaller than the size of the original data set. An augmented concept for coresets is the notion of *composable coresets* which have the following property: for a collection of sets, the approximate solution to the union of the sets in the collection can be obtained given the union of the composable coresets for the point sets in the collection. This notion was formally defined in a recent paper by Indyk et al [19]. In this paper, we augment the notion of composable coresets further, and introduce the concept of *mapping coresets*. A mapping coreset is a coreset with an additional mapping of points in the original space to points in the coreset. As we will see, this will help us solve balanced clustering problems for a wide range of objective functions and a variety of massive data processing applications, including streaming algorithms and MapReduce computations.

Roughly speaking, this is how a mapping coreset is used to develop a distributed algorithm for the balanced clustering problems: we first partition the data set into several blocks in a specific manner. We then compute a coreset for each block. In addition, we compute a mapping of points in the original space to points in the coreset. Finally, we collect all these coresets, and then solve the clustering problem for the union of the coresets. We can them use the (inverse) map to get back a clustering for the original points.

This idea is, for instance, directly applicable in the MapReduce framework [13] and gives an approximation algorithm in a constant number of rounds of MapReduce. A sample setting of parameters for a coreset based algorithm is as follows: each of $\sqrt{n/k}$ mappers receives $\sqrt{kn}$ points as input and produces a composable coreset of size k for this point-set. The produced coresets are then passed to a single reducer. The total input to the reducer, i.e., the union of the coresets, is of size at most $k\sqrt{n/k} = \sqrt{kn}$. The solution computed by the reducer for the union of the coresets is, by definition (since we have composable coresets), a good approximation to the original problem. Finally we use the mapping associated with the coreset to allocate each point in the orignal space to the corresponding point in the coreset that it was mapped to. Variants of this technique have been applied for optimization under MapReduce framework [20, 24, 5], however, none of the previous work needed to keep the mapping in addition to the composable coresets. This new ingredient is necessary for producing a valid output for balanced clustering problems. As we will elaborate later if we apply the technique sketched above, we may end up using a superlinear memory usage (because single-machine algorithms for capacitated problems often have quadratic memory requirement). To overcome this issue we will use the above idea multiple times in an iterative manner to compress the data even further and this process will result in an algorithm with $O(1)$ rounds of MapReduce (or passes of streaming computation).

1.1 Our contributions

In this paper, we introduce a framework for solving distributed clustering problems. Using the concept of mapping coresets as described above, our framework applies to *balanced* clustering problems, which are much harder than their unrestricted counterparts in many aspects.

The rough template of our results is the following: given a single machine α-approximation algorithm for a clustering problem (with or without balance constraints), we give a distributed algorithm for the problem that has an $O(\alpha)$ approximation guarantee. Our results also imply streaming algorithms for such clustering problems, using sublinear memory and constant number of passes.

More precisely, we consider balanced clustering problems with an ℓ_p objective. For specific choice of p, it captures the commonly used k-center, k-median and k-means objectives.

Our results are also very robust—for instance, bicriteria approximations (violating either the number of clusters or the cluster sizes) on a single machine can be used to give distributed bicriteria approximation algorithms, with a constant loss in the cost. This is particularly important for balanced versions of k-median and k-means, for which we know of constant factor approximation to the cost only if we allow violating one

MapReduce model		
Problem	Approximation	Rounds
L-balanced k-center	O(1)	O(1)
k-clustering in ℓ_p	$O(p)$	O(1)
L-balanced k-clustering in ℓ_p	(O(p),2)	O(1)
Streaming model		
Problem	Approximation	Passes
L-balanced k-center	O(1)	O(1)
k-clustering in ℓ_p	$O(p)$	O(1)
L-balanced k-clustering in ℓ_p	(O(p),2)	O(1)

Table 1: Our contributions, all results hold for $k < n^{1/2-\epsilon}$, for constant $\epsilon > 0$. We notice that for the L-balanced k-clustering (p) general we get a bicriteria optimization (we can potentially open $2k$ centers in our solutions).

of the constraints. (Moreover, mild violation might not be terribly bad in certain applications, as long as we obtain small cost.)

Finally, other than presenting the first distributed approximations for balanced clustering, our general framework also implies constant-factor distributed approximations for a general class of uncapacitated clustering problems (for which we are not aware of distributed algorithms with formal guarantees). We summarize our new results in Table 1.

Techniques While our framework is reminiscent of the divide and conquer coreset algorithms used in distributed clustering, balanced problems pose a novel challenge—keeping track of individual points (in some form, since we need to know the number of points in different clusters), while at the same time not storing them all. The definition of mapping coresets can also be thought of as a *succint representation*.

In other words, we can think of picking a small number of distinct locations, and moving each point in the input space to one of these locations. The succint representation is now the set of locations and their multiplicity. This kind of representation has been useful in other contexts, such as streaming algorithms (it is very much like the idea of Guha et al. [16]). However in streaming, the notion of 'composability' of these representations does not arise, while this is necessary for distributed computation!

Our main contribution is to construct the succint representation in a clever way so as to ensure composability. To this end, we rely on the classical k-median algorithm of Lin and Vitter [25], followed by a careful argument to ensure we preserve the cost and the balance of the solution.

1.2 Related work

Coresets: The notion of coresets has been introduced in [2]. Informally, a coreset for an optimization problem is a subset of the data with the property that solving the underlying problem on the subset gives an approximate solution for the original data. The notion is somewhat generic, and many variations of coresets exist. In this paper, we use the term coresets to refer to an augmented notion of coresets, referred to as "composable coresets". This term has been formally defined in a recent paper that applied the concept of composable coresets to develop MapReduce-based and streaming algorithms [19]. This notion has been also implicitly used in Section 5 of Agarwal et al. [2] where the authors specify composability properties of ϵ-kernels (a variant of coresets). The notion of (composable) coresets are also related to the concept of mergeable summaries that have been studied in the literature [1]. The main difference between the two is that aggregating mergeable summaries does *not* increase the approximation error, while in the case of coresets the error amplifies. As discussed before, the idea of using coresets has been applied either explicitly or implicitly in the streaming model [16, 2] and in the MapReduce framework [20, 24, 5, 19]. However, none

of the previous work applies these ideas for balanced clustering problems nor do they discuss the use of mapping coresets.

Distributed clustering. There has been a lot of work on designing efficient distributed algorithms for clustering problems in metric spaces. We will briefly summarize some results on the MapReduce and streaming models that are most relevant to our results.

A formal computation model for the MapReduce framework has been introduced by Karloff et al. [20]. The first paper that studied clustering problems in this model is by Ene et al. [14], where the authors prove that one can use an α approximation algorithm for the k-center or k-median problem to obtain a $4\alpha + 2$ and a $10\alpha + 3$ approximation respectively for the k-center or k-median problems in the MapReduce model. Subsequently Bahmani et al. [4] showed how to implement k-means++ efficiently in the MapReduce model. Finally, very recently, Balcan et al. [5] demonstrate how one can use an α approximation algorithm for the k-means or k-median problem to obtain coresets in the distributed (and MapReduce) setting. They however do not consider the balanced clustering problems or the general set of clustering problems with the ℓ_p objective function.

The literature of clustering in the streaming model is very rich, we will present a few of the most relevant results. The first paper to analyze clustering problem is by Charikar et al. [8] studying the k-center problem on the classic streaming setting. Subsquentially in a seminal paper Guha et al. [16] give the first single pass constant approximation algorithm to the k-median problem. Subsequently the memory requirements and the approximation factors of their result have been improved by Charikar et al. in [10]. The closely related facility location problem has been also studied, the first result in this context is by Indyk [18], who gave a $O(\log^2 \Delta)$ approximation streaming algorithm. The best currently known streaming algorithm is due to Lammersen and Sohler [23], and gives a $O(1)$-approximation for this problem.

Capaciated clustering. Capacitated (or balanced) clustering is a well studied area in approximation with many applications and challenging open questions. The early works in this area have been those on capacitated k-center, for instance Bar-Ilan et al. [7] and Khuller et al. [21]. For facility location, a series of works based on local search, culminating with Bansal et al. [6], have obtained constant factor approximation algorithms. However for other natural objectives such as k-median, no constant factor approximation algorithms are known. This has led to a study of many bicriteria approximation algorithms [9, 11].

Thus our results may be interpreted as saying that the capacity constraints may be a barrier to approximation, but are not a barrier to parallelizability.

This is also the reason our guarantees for all (except the k-center problem) are bicriteria.

1.3 Organization

In what follows, we start by giving some preliminary definitions before explaining our general framework in Section 3. More details on how the framework can be implemented in MapReduce and Streaming models as well as the challenges we face in doing so are given in Sections 3.2 and 3.3, respectively. Section 4 delves more deeply into proving the soundness of the framework. In particular it provides details of the algorithm for finding the coreset as well as bounds the approximation loss incurred in different steps. Finally, Section 5 discusses several sequential algorithms that we can plug in as a black box in our algorithm and the results that we obtain.

2 Preliminaries

In all the problems we study, we will denote by (V, d) the metric space we are working with. We will denote $n = |V|$, the number of *points* in V. We will also write d_{uv} as short hand for $d(u, v)$. Given points u, v, we assume we have an oracle access to d_{uv}; note that this assumption is not trivial: if d is a metric on a graph and we have a distributed setting, then finding $d(u, v)$ between two points u, v on one machine might need access to points on other machines. However in most applications, we have points in geometric spaces (where distances can be computed knowing u, v), thus we will ignore this issue.

Formally, a clustering $\mathcal{C}$ of a set of points V is a collection of sets $C_1, C_2, \ldots, C_r$ which partition V. Each cluster C_i has a center v_i, and we define the 'ℓ_p cost' of this clustering as

$$\mathsf{cost}_p(\mathcal{C}) := \left(\sum_i \sum_{v \in C_i} d(v, v_i)^p \right)^{1/p}. \tag{1}$$

When p is clear from the context, we will simply refer to this quantity as the cost of the clustering and denote it $cost(\mathcal{C})$.

Let us now define the L-balanced k-clustering problem with ℓ_p cost.

Definition 1 (L-balanced k-clustering (p))**.** *Given (V, d) and a size bound L, find a clustering $\mathcal{C}$ of V which has at most k clusters, at most L points in each cluster, and cluster centers $v_1, \ldots, v_k$ so as to minimize $\mathsf{cost}_p(\mathcal{C})$, the ℓ_p cost defined in Eq.* (1)*.*

As mentioned earlier, the version of the problem without the balanced constraint (or $L = n$) is the ℓ_p k-median problem. With the balanced constraint, the case $p = 1$ is the notorious capacitated k-median, which we discussed earlier, and $p = \infty$ is also known as the capacitated k-center problem (with uniform capacities).

Definition 2 (Mapping and mapping cost)**.** *Given a multiset S and a set V, we call a bijective function $f : V \to S$ a mapping from V to S and we define the cost of a mapping as $\sum_{v \in V} d(v, f(v))^p$.*

Definition 3 (Clustering and optimal solution)**.** *Given a clustering problem $\mathcal{P}$, we define $OPT_{\mathcal{P}}$ as the cost of the optimal solution to $\mathcal{P}$.*

3 Our framework

The main idea behind our new results is the definition of a new family of coresets that helps in dealing with balanced clustering.

Definition 4 (δ-mapping coreset)**.** *Given a set of points V, a δ-mapping coreset for a clustering problem $\mathcal{P}$ consists of a multiset S with elements from V, and a mapping from V to S such that the total cost of the mapping is upper bounded by $(1 + \epsilon)\delta OPT^p_{\mathcal{P}}$, for any constant $\epsilon > 0$. We define the size of a δ-mapping coreset as the number of distinct elements in S.*

Remark. A δ-mapping coreset can thus be thought of as a subset S^* of V, along with some multiplicites (or weights), and a mapping $g : V \to S^*$, such that the weight of a point $s \in S^*$ is $|g^{-1}(s)|$. It is thus a somewhat stronger notion than coresets in earlier works [5, 15], which have weighted points, but which do not arise from a mapping of the original points. This condition is crucial for handling bounds on cluster sizes. Another difference is that the definition only requires a bound on the mapping cost – while standard notions require that $OPT_{\mathcal{P}}$ must be the same for the original set and the coreset. This turns out to be a *consequence* of our definition (see Section 3.1).

In our algorithms we will use a strengthen version of δ-mapping coreset called composable δ-mapping coreset.

Definition 5 (Composable δ-mapping coreset)**.** *Given sets of points $V_1, V_2, \ldots, V_m$, and corresponding δ-mapping coresets $S_1, S_2, \ldots, S_m$, the coresets are said to be* composable *if we have that $\cup_i S_i$ (taking union as a multiset) is a $2^p\delta$-mapping coreset for $\cup_i V_i$.*

Remark. Note that the reason it is not trivial that mapping coresets *compose* as above, is that in the definition, we compare the mapping cost to the cost of $OPT_{\mathcal{P}}$ on the set of points, so we will need to show that the optimum is not too small for $\cup_i V_i$.

In fact, our main theorem will say that for the family of clustering problems defined earlier, it is possible to construct a composable δ-mapping coreset. More formally:

Theorem 1. *For any point set V and any $L, k, p \geq 1$, there is an algorithm that produces a composable 2^p-mapping coreset for the L-balanced k-clustering (p) problem. Furthermore the size of this coreset is $\tilde{O}(k)$, and the algorithm uses space that is quadratic in $|V|$.*

Note. We defined composability for a collection of coresets. The theorem should be interpreted as: suppose we use the algorithm (from the theorem) on any vertex sets $V_1, \ldots, V_m$ (any m), the coresets thus obtained are composable.

In the next subsections we describe a framework to cluster points in distributed settings using Theorem 1, and the notions developed so far.

3.1 Clustering via composable δ-mapping coreset

To use composable δ-mapping coreset for clustering, we first show that a clustering computed on points in S can be mapped back to a good clustering on the initial set of points V and vice-versa. This is exactly what we show in the next two theorems:

Theorem 2. *Let $\mathcal{C}$ be any clustering of V, and let $\mathcal{C}'$ denote the clustering of S obtained by applying a bijection f to the clustering $\mathcal{C}$. Then there exists a choice of centers for $\mathcal{C}'$ such that*

$$\mathsf{cost}(\mathcal{C}')^p \leq 2^{2p-1}(\mathsf{cost}(\mathcal{C})^p + \mu_{total}),$$

where μ_{total} denotes $\sum_{v \in V} d(v, f(v))^p$.

Furthermore, the above lemma holds also in the opposite direction:

Theorem 3. *Let $\mathcal{C}'$ be a clustering of S, and let $\mathcal{C}$ denote the clustering of V obtained by applying a bijection f^{-1} to the clustering $\mathcal{C}'$. Then there exists a choice of centers for $\mathcal{C}$ such that*

$$\mathsf{cost}(\mathcal{C})^p \leq 2^{2p-1}(\mathsf{cost}(\mathcal{C}')^p + \mu_{total}).$$

Additionally, the new set of centers for clustering $\mathcal{C}$ can be found in time linear in the size of the clustering.

Preserving balanced property. The two theorems allow us to move back and forth (algorithmically) between clusterings of V and S as long as there is a small-cost mapping. Furthermore, since the clusterings are essentially the same (f is a bijection), we have the property that if the clustering was balanced in V, the corresponding one in S will be balanced as well, and vice versa.

Let us now see how exactly we use the theorems. If we have a bijection f for which μ_{total} is bounded and an α approximation for a clustering problem $\mathcal{P}$, then we can use the Theorem 2 to show that in S, there is at least a clustering whose cost elevated to the p-th power is smaller than $2^{2p-1}(\mathsf{cost}(\mathcal{C})^p + \mu_{\text{total}})$, then we can use the approximation algorithm to get a $2^{2p-1}\alpha^p(\mathsf{cost}(\mathcal{C})^p + \mu_{\text{total}})$ upper bound in S. Finally, using Theorem 3, we can map back the clusters from S to V and get a an upper bound to the p power of the clustering cost of $2^{2p-1}(2^{2p-1}\alpha^p(\mathsf{cost}(\mathcal{C})^p + \mu_{\text{total}}) + \mu_{\text{total}})$. But now using Theorem 1, we have a function f such that $\mu_{\text{total}} = 2^p\mathsf{cost}(\mathcal{C})^p$. So plugging this into the bound above, and after some algebraic manipulations we obtain that the cost of the final clustering is bounded by $32\alpha\mathsf{cost}(\mathcal{C})$.

An important property of our composable 2^p-mapping coreset is that its size is $\tilde{O}(k)$, so we can use it to compress the input instance (using the map).

Definition 6 (Compressed representation)**.** *Given an instance (V, d) for a k-clustering problem, we say that the instance has a compressed representation if it is possible to describe V using only Y points, with $|Y| \in \tilde{O}(k)$, and their multiplicity.*

Our distributed framework is as follows: if the input is too large to fit in a single machine we first split the input in m chunks. Then we can compute a composable 2^p-mapping coreset for each of the chunks. Subsequently, we can use their composability property to merge all the coresets and compute a clustering there. A crucial technical point here is that the clustering algorithm should work when the points have multiplicities (this is true of all algorithms we are aware of). Formally, we call these *space efficient* algorithms.

Definition 7 (Space-efficient algorithm)**.** *Given an instance (V, d) for a k-clustering problem with a compressed representation Y, a sequential α-approximation algorithm is called space-efficient if the space used by the algorithm (not considering the space used to write the output) is in $O(|Y|^2 \cdot poly(k))$.*

Our definition allows a fairly large dependence on k, though in all cases we consider, the poly() will be constant or linear – we allow such slack because in many applications, it is reasonable to think of k as much smaller than n.

The final step in our framework is to move back from a clustering of S to one of V, i.e., reverse the composable 2^p-mapping coreset function f (again using m machines). This overall framework allows us to compute the clustering of the large dataset using a small amount of memory by losing a constant factor in the approximation.

Multiple compositions. Furthermore note that if we have a very large input or very limited amount of memory we can apply our composable 2^p-mapping coresets iteratively multiple times (as we illustrate in two settings below). In particular if we apply it t times, then using a sequential α approximation algorithm at the bottom-most 'level', we will obtain a $32^t\alpha$ approximation algorithm for the full instance. We will now illustrate the framework in detail in MapReduce and the Streaming setting.

3.2 Mapping Coresets for Clustering in MapReduce

We start by recalling the main aspects of the MapReduce model introduced by Karloff et al. [20].

The model has two main restrictions, one on the total number of machines and another on the memory available on each machine. In particular, given an input of size N, and a sufficiently small $\gamma > 0$, in the model there are $N^{1-\gamma}$ machines, each with $N^{1-\gamma}$ memory available for the computation. As a result, the total amount of memory available to the entire system is $O(N^{2-2\gamma})$. In each round a computation is executed on each machine in parallel and then the outputs of the computation are shuffled between the machines.

In this model the efficiency of an algorithm is measured by the number of the 'rounds' of MapReduce in the algorithm. A class of algorithms of particular interest are the ones that run in a constant number of rounds. This class of algorithms are denoted $\mathcal{MRC}^0$.

The high level idea is to use coreset construction and a sequential space-efficient α-approximation algorithm (as outlined above). Unfortunately, this approach does not work as such in the MapReduce model because both our coreset construction that a space-efficient algorithm need memory quadratic in the input size (on each machine). Therefore, depending on the k, we perform multiple 'levels' of our framework, as described at the end of Section 3.1.

Given an instance (V, d), the MapReduce algorithm proceeds as follows:

1. Partition the points arbitrarily into $2n^{(1+\gamma)/2}$ sets.
2. Compute the composable 2^p-mapping coreset on each of the machines (in parallel) to obtain f and the multisets $S_1, S_2, \ldots, S_{2n^{(1+\gamma)/2}}$, each with roughly $\widetilde{O}(k)$ distinct points.
3. Partition the computed coreset again into $n^{1/4}$ sets.
4. Compute composable 2^p-mapping coresets on each of the machines (in parallel) to obtain f', and multisets $S'_1, S'_2, \ldots, S'_{n^{1/4}}$, each with $\widetilde{O}(k)$ distinct points.
5. Merge all the $S'_1, S'_2, \ldots, S'_{n^{1/4}}$ on a single machine and compute a clustering using the sequential space-efficient α-approximation algorithm.
6. Map back the points in $S'_1, S'_2, \ldots, S'_{n^{1/4}}$ to the points in $S_1, S_2, \ldots, S_{2n^{(1+\gamma)/2}}$ using the function f'^{-1} and obtain a clustering of the points in $S_1, S_2, \ldots, S_{2n^{(1+\gamma)/2}}$.
7. Map back the points in $S_1, S_2, \ldots, S_{2n^{(1+\gamma)/2}}$ to the points in V using the function f^{-1} and thus obtain a clustering of the initial set of points.

Note that if $k < n^{1/4-\epsilon}$, for constant $\epsilon > \gamma$, at every step of the MapReduce, the input size on each machine is bounded by $n^{(1-\gamma)/2}$ and thus we can run our coreset reduction and a space-efficient algorithm (in which we think of the poly(k) as constant – else we need minor modification).

Furthermore if $n^{1/4-\epsilon} \leq k < n^{(1-\epsilon)/2}$, for constant $\epsilon > \gamma$, we can exploit the trade-off between number of rounds and the reduction on the size of the instance in each round to have a constant approximation to the L-balanced k-clustering (p) problem in a constant number of rounds. This is because if k is larger than $n^{1/4-\epsilon}$, we can use different rounds in which we compute coresets of the input to compress the data from $n^{(1-\gamma)/2}$ gradually down to $\tilde{O}(k)$. More specifically, using this technique in every round we can reduce the number of partitions by a factor $\tilde{O}\left(\frac{n^{\frac{(1-\gamma)}{2}}}{k}\right)$. Thus for a $k < n^{\frac{1}{2}-\epsilon}$, the number of MapReduce steps t required to reduce the problem to a single-machine problem is equal to the minimum t such that

$$\left(\frac{n^{\frac{(1-\gamma)}{2}}}{k \log n}\right)^t \geq 2n^{\frac{(1+\gamma)}{2}}.$$

We are now ready to state our main theorem:

Theorem 4. *Given an instance (V, d) for a k-clustering problem, with $|V| = N$ and a sequential space-efficient α approximation algorithm to the (L-balanced) k-clustering (p) problem, there exists a MapReduce algorithm that runs for a constant number of rounds and obtains a $O(\alpha)$ approximation for the (L-balanced) k-clustering (p) problem, for $L, p \geq 1$ and $0 < k < n^{(1-\epsilon)/2}$, for constant $\epsilon > \gamma$.*

The previous theorem combined with the results of Section 5 gives us the following three corollaries:

Corollary 5. *There exists a MapReduce algorithm that runs for a constant number of rounds and obtains an $O(1)$ approximation for the L-balanced k-center problem, for every $0 < k < n^{1-\epsilon/2}$, for some small constant $\epsilon > \gamma$.*

Corollary 6. *There exists a MapReduce algorithm that runs for a constant number of rounds and obtains an $O(p)$ approximation for the k-clustering (p) problem, for every $1 < p \leq \log n$ and $0 < k < n^{1-\epsilon/2}$, for some small constant $\epsilon > \gamma$.*

Corollary 7. *There exists a MapReduce algorithm that runs for a constant number of rounds and obtains an $(O(p), 2)$ bicriteria approximation for the L-balanced k-clustering (p) problem, for every $1 < p \leq \log n$ and $0 < k < n^{1-\epsilon/2}$, for some small constant $\epsilon > \gamma$.*

Note that Corollary 7 gives a bicriteria approximation (in the worst case we can open up to $2k$ center) because also the best known sequential approximation algorithms for those problems are bicriteria.

3.3 Balanced clustering in the streaming model

We now observe that δ-mapping coresets can also be used to obtain streaming algorithms for balanced clustering. The idea is to process the stream in chunks, keep track of mapping coresets in each chunk, and then work with their union. The issue though, is that the bijection between the original points and the coreset cannot be stored (since it takes too much memory)[2]. However if we perform a second pass over the data (after finding the clustering on the coreset), we can recompute the mapping, look up the cluster that the mapped vertex belongs to, and return the cluster index. This gives a two-pass streaming algorithm. We give the details below. We also note that the algorithm bears close resemblance to that of Guha et al. [16].

Let m be a parameter we will decide on shortly, and let the stream consist of points $v_1, v_2, \ldots, v_n$ arriving in that order.

1. Process the stream in m batches, $V_1, V_2, \ldots, V_m$ (each contains n/m of the original points).

[2]In a distributed system, we could store across several machines.

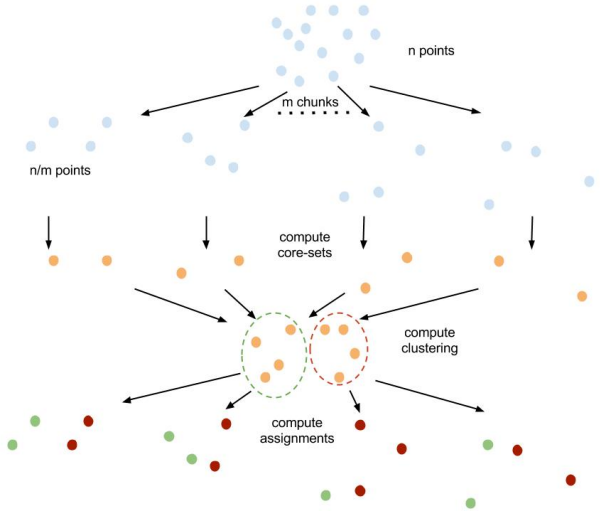

Figure 1: To process an input that does not fit in a single machine, we split into m parts, compute composable mapping coresets for each part, and aggregate them. We then compute a solution to this aggregate and map the clustering back to the input.

2. For each i, compute a composable δ-mapping coreset S_i of V_i.

3. Compute the optimum solution to the k-clustering problem on $\cup_i V_i$. It consists of a set of k centers T, and a set of k integers for each point s in $\cup_i S_i$ (which store how many of the original points mapped to s are assigned to each of the centers in T). Let us call this array of k integers g_s.

4. Take a second pass through the stream; For each point v in the stream, find the closest point in the corresponding S_i (say it is some s). Then assign v to the cluster corresponding to the first non-zero g_s, and decrement that array-entry in g_s.

Note that this procedure outputs the index of the cluster for each point. If we also want to output cluster centers, it is easy to keep track of it using another array – the key is that all the points v that are mapped to a location s are "equivalent" in later stages of the algorithm.

Memory usage. Let us observe how much memory we used: we kept track of $\widetilde{O}(mk)$ points in the form of $\cup_i V_i$. Each V_i has size $O(n/m)$. Now if we use a space efficient algorithm and our mapping core-set we use quadratic space. Thus if we pick $mk = n/m$, or $m = \sqrt{n/k}$, the total space complexity is $\widetilde{O}(nk)$. Note that this is super linear. But this is precisely the issue we faced in the MapReduce framework, which we got around by using 'many rounds'.

In the streaming framework, this simply means that we perform a more steps of aggregation. For instance, suppose we write $m = p \cdot q$, and break up step (2) above as follows: we compute $S_1, S_2, \ldots S_p$ as before, and then compute a δ-mapping coreset of their union (and store only this), and continue aggregating chunks of p of the S_i. This means the space used is roughly quadratic in $\widetilde{O}(qk + pk + (n/m))$. Thus if we pick $p = q = (n/k)^{1/3}$ and $m = (n/k)^{2/3}$, we have the space bound to be $\widetilde{O}(n^{2/3}k^{4/3})$, which is sub-linear when $k < n^{1/4-\epsilon}$, for constant $\epsilon > 0$. More generally if we have $k < n^{1/2-\epsilon}$ and a space-efficient α approximation algorithm for a (L-balanced) k-clustering (p) problem, then using roughly $O(1/\epsilon)$ levels, we can obtain a $O(\alpha)$ approximation to the same problem in the streaming model.[3] More formally:

[3]The point we glossed over is the following: when we consider multiple levels, we cannot tell in the second pass, while considering u, which vertex in the final coreset u is mapped to. This is dealt with by recomputing a mapping right after step (2) (by doing another pass and assigning vertices to the closest in the coreset). In the process, the multiplicities of vertices

Theorem 8. *Given an instance (V, d) for a k-clustering problem, with $|V| = n$ and a sequential space efficient α approximation algorithm to the (L-balanced) k-clustering (p) problem, there exists a streaming algorithm for the problem which computes an $O(\alpha)$ approximation using sublinear space and three passes over the input, for $L, p \geq 1$ and $0 < k < n^{(1-\epsilon)/2}$, for constant $\epsilon > 0$.*

Using the previous theorem combined with the results of Section 5 we can obtain similar corollaries to the one shown in subsection 3.2.

4 Coresets and Analysis

We now come to the proof of our main result—Theorem 1. We show the algorithm to construct coresets, then prove some of the theorems crucial for our framework, and show how the various results fit in to prove the theorem.

4.1 Constructing composable coresets

Suppose we are given a set of points V. We first show how to select a set of points S that are *close* to each vertex of the V. We then form the mapping f (in the definition of a mapping coreset), and this decides the multiplicity of the points. The selection of S uses a modification of the algorithm of Lin and Vitter [25] for k-median.

Consider a solution (x, y) to the following linear programming (LP) relaxation:

$$\begin{aligned} \min \sum_u \sum_v d(u,v)^p x_{uv} \qquad & \text{subject to} \\ \sum_v x_{uv} = 1 \qquad & \text{for all } u \\ x_{uv} \leq y_v \qquad & \text{for all } u, v \\ \sum_u y_u \leq k & \\ 0 \leq x_{uv}, y_u \leq 1 \qquad & \text{for all } u, v. \end{aligned}$$

In the above algorithms, we can always treat $p > \log n$, and in particular the case $p = \infty$, as $p = \log n$. This introduces only negligible error in our computations but make them tractable. More specifically, when working with $p = \log n$, the power operators do not increase the size of the input by more than a factor $\log n$.

Also, for $p > 1$, it is important that we are not minimizing $\sum_u \left(\sum_v d(u,v) x_{uv} \right)^p$ since the objective is not linear in that case. However, note that the integer program corresponding to the above exactly captures the (unbalanced) version of k-median with ℓ_p connection costs.

Rounding We do simple randomized rounding (with a logarithmic boost): round each y_u to 1 with a probability equal to $\min\{1, y_u(4 \log n)/\epsilon\}$. Let us denote this probability by y'_u, and the set obtained by S.

Lemma 9. *With probability $(1 - 1/n)$, the set S of selected centers satisfies the following properties.*

1. *Each vertex has a relatively close selected center. In particular, for every $u \in V$, there is a center opened at distance at most $\left[(1+\epsilon) \sum_v d(u,v)^p x_{uv} \right]^{1/p}$.*
2. *Not too many centers are selected; i.e., $|S| < \frac{8k \log n}{\epsilon}$.*

in the final coreset change (but not the distinct points), and then perform step (3). A straightforward argument shows that the costs remain bounded. We do not get into this technical issue, but note that it is the reason we have *three* passes in the theorem statement.

of Lemma 9. Define $C_u = \sum_v d(u,v)^p x_{uv}$, and let $B_u = B(u, [(1+\epsilon)C_u]^{1/p})$ be the ball we are interested in. We first claim that $\sum_{v \in B_u} y_v \geq \epsilon/2$. This is by Markov inequality: at least $\epsilon/2$ of the contribution (in terms of x_{uv}) in the sum $\sum_v d(u,v)^p x_{uv}$ must come from vertices v such that $d(u,v)^p < (1+\epsilon)C_u$. Thus since $y_v \geq x_{uv}$, we have the desired claim.

Now, the probability that we do not pick any vertex from B_u is $\prod_{v \in B_u}(1 - y'_v)$. The product is zero if $y'_v = 1$ for any $v \in B_u$. If no term is zero, we can upperbound the product by $\exp(-\sum_v y'_v) \leq \exp(-\sum_v 4y_v \log n/\epsilon) \leq \exp(-2\log n) < 1/n^2$. Thus, taking a union bound over all u, we obtain the desired result.

The second part, i.e., the bound on the size of S follows trivially from a Chernoff bound. □

Mapping and multiplicity. Finally, each vertex $v \in V$ is mapped to the center closest to it, i.e., $f(v) = \arg\min_{s \in S} d(v,s)$. If m_s points in V are mapped to some $s \in S$, we set its multiplicty to m_s. This defines a bijection (also called f, mildly abusing notation) from V to the resulting multiset.

4.2 Comparing costs

We now wish to consider a clustering $\mathcal{C}$ of V, and compare the cost to the same clustering (after applying the bijection) of S. This will end up proving Theorems 2 and 3.

Remark. Before we begin, we remark that in the theorems, it is important that we leave open the choice of the cluster centers. It is easy to construct examples where picking f_{v_i} as the centers (where v_i are the centers in $\mathcal{C}$) results in a large cost.

During the proofs in this section, we often use the "triangle inequality" (with a scaling factor) for ℓ_p by which we refer to the following. For $p \geq 1$ and three points u, v, w, we have

$$d(u,v)^p \leq 2^{p-1}[d(u,w)^p + d(w,v)^p].$$

What looks more like the normal triangle inequality, which we also use in the proofs, is

$$\sqrt[p]{A+B} \leq \sqrt[p]{A} + \sqrt[p]{B},$$

holding for $p \geq 1$ and $A, B \geq 0$.

of Theorem 2. Consider some cluster $C_i \in \mathcal{C}$, and its center v_i. Let $C'_i = \{f_v : v \in C_i\}$ (i.e., the corresponding cluster in $\mathcal{C}'$. Now let s be the point in C'_i that is closest to v_i (it need not be f_{v_i}, as remarked above). We claim that choosing s to be the center of C'_i gives a low cost to the cluster.

Thus we are interested in the quantity $\sum_{s' \in C'_i} d(s,s')^p$, which can be rewritten as $\sum_{v \in C_i} d(s, f_v)^p$. For $p \geq 1$ we can write the triangle inequality with a scaling factor to obtain

$$d(s, f_v)^p \leq 2^{p-1}[d(s,v_i)^p + d(v_i, f_v)^p]$$

and then we use the choice of s to get

$$\leq 2^{p-1} \cdot 2d(v_i, f_v)^p$$

and finally another application of triangle inequality gives

$$\leq 2^{2p-1}[d(v_i,v)^p + d(v, f_v)^p].$$

Summing over the clusters and all points in them, we get

$$\mathsf{cost}(\mathcal{C}')^p \leq 2^{2p-1}(\mathsf{cost}(\mathcal{C})^p + \mu_{\text{total}})$$

as promised. □

In particular, applying triangle inequality to the theorem gives $\mathsf{cost}(\mathcal{C}') \leq 4(\mathsf{cost}(\mathcal{C}) + \sqrt[p]{\mu_{\text{total}}})$, and for the special case of $p = 1$ we have $\mathsf{cost}(\mathcal{C}') \leq 2(\mathsf{cost}(\mathcal{C}) + \mu_{\text{total}})$.

Theorem 3 proceeds in the opposite direction (moving from a clustering in S to one in V). Since the above argument above only depends on the bijection, we can use it with the roles of V, S interchanged. It also gives an algorithm to convert a clustering in S to a clustering in V (which we need). As for computing the centers, for each cluster we pick the new center to be the point in the cluster closest to the old center, thus it can be found in linear time (knowing f^{-1}) for each cluster, hence in linear time in total.

4.3 Composability of the coresets

We now come to the crucial step, the proof of composability for the mapping coresets constructed earlier, i.e., Theorem 1.

To show this, we consider any vertex sets $V_1, V_2, \ldots, V_m$, and mapping coresets $S_1, S_2, \ldots, S_m$ obtained by the rounding algorithm above. We have to prove that the total moving cost is at most $(1+\epsilon)2^p OPT_{\mathcal{P}}$, where the optimum value is for the instance $\cup_i V_i$. We denote by $LP(V_i)$ the optimum value of the linear program above, when the set of points involved is V_i. Finally, we write $\mu_v := d(v, f_v)^p$, and $\mu_{\text{total}} := \sum_{v \in V} \mu_v$. The following lemma is simple.

Lemma 10. *Let LP_i denote the objective value of the optimum solution to $LP(V_i)$, i.e., the LP relaxation written earlier when only vertices in V_i are considered. Then we have $\mu_{total} \leq (1+\epsilon)\sum_i LP_i$.*

Proof. The proof follows directly from Lemma 9 and the definition of f (that it maps to the closest point in S_i). □

Lemma 11. *In the notation above, we have $\sum_i LP_i \leq 2^p \cdot LP(V)$.*

Proof. The lemma will follow if we show that for every i, the quantity LP_i is at most 2^p times the contribution of vertices in V_i to the objective of $LP(V)$ (because $LP(V)$ is simply a sum of these contributions since V_i is a partition).

To show this, we will construct a solution to $LP(V_i)$ using a solution to $LP(V)$ with the cost bounded as above. This is done as follows. Fix some i and consider an optimal solution (x, y) to $LP(V)$. If we have $y_u = 0$ for all $V \setminus V_i$, then the same solution, restricted to $u \in V_i$ (in both x and y), works for $LP(V_i)$. Otherwise we redistribute y-'mass' from some $u \in V \setminus V_i$ to some $u' \in V_i$—this is done by picking the vertex $u' \in V_i$ that is closest to u, and setting $y_{u'} := \max\{y_u + y_{u'}, 1\}$, (the second term in the sum is the 'current' value of $y_{u'}$), and setting $y_u = 0$. Note that this process does not increase $\sum_u y_u$. Further, for all vertices $w \in V_i$, we set $x_{wu'} := \max\{x_{wu} + x_{wu'}, 1\}$.

Now repeat this process to remove all y-mass from $u \in V \setminus V_i$, and modify x appropriately. It is easy to see that $\sum_u y_u \leq k$, and further, restricting the solution (both x, y) to the indices in V_i satisfies all the conditions in $LP(V_i)$.

It only remains to bound the cost of the solution. The key here is to note that when we increment $x_{wu'}$ above, the cost of the solution could increase by an amount $x_{wu} d(w, u')^p$. Now since u' is the closest point in V_i to u, it must be that $d(w, u')^p \leq 2^{p-1}[d(w, u)^p + d(u, u')^p] \leq 2^p d(w, u)^p$. But $x_{wu} d(w, u)^p$ is precisely the cost for the fractional assignment $w \to u$ in the original LP solution.

Thus it follows that the value LP_i is at most 2^p times the contribution of the vertices in V_i to the objective in $LP(V)$, proving the lemma. □

These two lemmas imply that the total mapping cost is at most $(1+\epsilon)2^p OPT_{\mathcal{P}}$, because $LP(V)$ is clearly $\leq OPT_{\mathcal{P}}$.

4.4 Wrapping up

Now suppose we have an α-approximation algorithm, which we want to use as a subroutine in our algorithm. Starting with the optimal solution $\mathcal{C}$ (to the original instance) of cost $\mathsf{cost}(\mathcal{C})$ we construct a second instance using the coreset. Theorem 2 guarantees a solution $\mathcal{C}'$ for this instance such that $\mathsf{cost}(\mathcal{C}')^p \leq 2^{2p-1}(\mathsf{cost}(\mathcal{C})^p +$

μ_{total}). We apply the α-approximation algorithm here to obtain $\mathcal{C}''$ such that $\mathsf{cost}(\mathcal{C}'')^p \leq \alpha^p 2^{2p-1}(\mathsf{cost}(\mathcal{C})^p + \mu_{\text{total}})$. By Theorem 3, $\mathcal{C}''$ can be mapped to a solution $\mathcal{C}'''$ of the original instance where

$$\begin{aligned}
\mathsf{cost}(\mathcal{C}''')^p &\leq 2^{2p-1}\left[\alpha^p 2^{2p-1}(\mathsf{cost}(\mathcal{C})^p + \mu_{\text{total}}) + \mu_{\text{total}}\right] \\
&= 2^{4p-2}\alpha^p \mathsf{cost}(\mathcal{C})^p + 2^{2p-1}(2^{2p-1}\alpha^p + 1)\mu_{\text{total}} \\
&\leq \mathsf{cost}(\mathcal{C})^p\left[2^{4p-2}\alpha^p + 2^{3p-1}(1+\epsilon)(2^{2p-1}\alpha^p + 1)\right] \\
&\leq \mathsf{cost}(\mathcal{C})^p\left[2^{4p-2}\alpha^p + 2^{3p-1}2^{2p}\alpha^p\right] \\
&\leq \mathsf{cost}(\mathcal{C})^p \alpha^p 2^{5p}.
\end{aligned}$$

This yields $\mathsf{cost}(\mathcal{C}''') \leq 2^5 \cdot \alpha \cdot \mathsf{cost}(\mathcal{C})$.

5 Solving on a single machine

Since our framework essentially reduces distributed computation to sequential computation on a smaller (multiset) instance, we will review the known algorithms for balanced k-clustering. For the k-center objective, the following is known [21] (this is called the uniform capacity version). We will see in Section 5.2 that this algorithm can be adapted to the case in which we have a multiset instance.

Theorem 12. *(See [21]) There exists a factor* 6 *approximation for the capacitated k-center problem when each of the centers has a capacity of L, for some L.*

Now what about general ℓ_p? As we mentioned earlier, this is a notorious open problem, even for uniform capacities and $p = 1$. Thus as in earlier works, we resort to bicriteria approximations. Here we can relax two constraints – (a) the bound k on the number of centers, and (b) the bound L on the size of the clusters. In practice, small violations in both these quantities are often acceptable. We will choose (a) above,[4] and show how to obtain a constant factor approximation, using at most $2k$ centers. We will call this an $(O(1), 2)$-bicriteria approximation.

Theorem 13. *Suppose there is an α approximation algorithm for the unconstrained (no balance constraint) k-clustering (p) problem. Then there exists a $(2\alpha, 2)$ bicriteria approximation for L-balanced k-clustering (p).*

The rest of the section is organized as follows: we first give a proof of Theorem 13 (which then results in giving bicriteria algorithms for all k-clustering problems we consider). Then, we show how to adapt the k-center algorithm of [21, 3] to make it space efficient (Definition 7). This step is quite general, and we believe most LP based algorithms should be adaptable in this way.

5.1 Bicriteria approximation

Let us now prove Theorem 13. The idea is very simple: to solve the unconstrained problem approximately, then open new centers "if necessary", and finally prove that we do not open too many centers. We now give the formal proof.

Proof. Given an instance (V, d, k, L) of L-balanced k-clustering (whose optimum value we denote OPT), first solve the unconstrained problem, and obtain an α-approximate solution. Clearly this has a cost at most $\alpha \cdot \text{OPT}$. Now if each cluster in this solution has at most L points, we are done – the same solution works for the L-balanced version.

Otherwise, we open new centers as follows. Consider some cluster C in the solution, that has a center u. Suppose there are n_C points in the cluster, and that $n_C > L$ (else we do not modify this cluster). Our new solution will break this cluster into $\lceil n_C/L \rceil$ clusters. The centers of the new clusters will be the $\lceil n_C/L \rceil$

[4] Working with (b) is typically harder – but it is not clear why this is more 'valid' in practice; Further, (a) allows for clean and distributed algorithms.

points in C that are closest to u. The rest of the points are assigned to the centers arbitrarily, so as to satisfy the L-balanced requirement. This choice of centers ensures that for every $v \in C$ that is not picked as a center, its new center u' satisfies $d(u, u') \leq d(u, v)$, which implies that $d(v, u') \leq 2d(v, u)$.

Thus in the new objective, we have $\sum_v d(v, u')^p \leq \sum_v 2^p d(v, u)^p$. Thus after taking pth root, the new objective value is at most 2αOPT. It just remains to bound the number of clusters. This is also easy: the new solution has $\sum_C \lceil n_C/L \rceil < \sum_C n_C/L + 1$ new clusters. Since there are $\leq k$ initial clusters (so the sum has $\leq k$ terms), the number of clusters is bounded by $k + (\sum_C n_C)/L$. The latter sum is clearly at most n/L, which must be at most k, because we start with the assumption that it is possible to cluster the n input points into k clusters of size at most L. This shows the bicriteria bound. □

Compressed instances. The caveat in our setting is that we have a succint instance (the set of points is a multiset), and we wish to obtain a space efficient algorithm (Defn. 7). It is easy to see that Theorem 13 works as is, except that we need an α approximation for the *vertex weighted* version of unconstrained k-clustering to start with.

Unconstrained algorithm. Theorem 13 above starts with a constant factor approximation for (weighted, unconstrained) k-clustering (p). While this is well known for $p = 1, \infty$, the best known bound for general p is via a swap-based local search algorithm, with an approximation factor of $O(p)$, due to Gupta and Tangwongsan [17].

We should note that their algorithm (as stated) does not handle weights. However it is easy to see directly that swap-based algorithms can be made space efficient (by working with the compressed instance). Suppose we had points at N locations each with a certain multiplictiy. Then we can keep an array of size N which maintains the current number of centers open at a location. Now when deciding to swap, we can simply keep track of the number of points 'available' at a location (multiplicity minus the current number of centers), and consider it in the swap procedure. (However, note that the optimum will not gain by picking multiple points at the same location – since the problem is unconstrained.)

5.2 Space efficient algorithm for k-center

The k-center algorithm of Khuller et al. [21] proceeds as follows. Guess the optimum R (this is the radius of each cluster, since we have k-center). Now we can assume that if we consider the graph on the points in which there is an edge (u, v) iff $d(u, v) \leq R$, then the whole graph is connected (else we can work with the two different subgraphs, since no assignments can be made across the subgraphs). Under this assumption, they consider a feasible solution to the following linear program.

$$\begin{aligned}
\sum_v x_{uv} &= 1 && \text{for all } u \\
x_{uv} &\leq y_v && \text{for all } u, v \\
\sum_u y_u &\leq k \\
\sum_u x_{uv} &\leq L \cdot y_v && \text{for all } v \\
0 \leq x_{uv}, y_u &\leq 1 && \text{for all } u, v.
\end{aligned}$$

They obtain a factor 6 approximation by rounding this linear program. Let us now see the issues involved in making this algorithm space efficient. This means we start with a succint instance, say with N locations, each with some multiplicity, and a given k (number of clusters, which will be $\leq N$ in our framework). The goal is to obtain an approximate solution using only a small amount of memory (which is in terms of N, k). Our result will be the following:

Theorem 14. *There exists a factor 8 approximation algorithm for k-center with capacities which works with a succint instance with N locations, that uses memory $O(N^2 k)$.*

We will only sketch the proof, since it is not the main subject of the paper. We note that we do not know how to make natural variants of the LP(using N^2 memory) have small integrality gap. The key difficulty is that in the optimum solution, a center could be opened many times, and it is not quite clear how to capture this directly in the LP.

Our idea is the following: suppose the locations u have multiplicities m_u. We first move to the "client-facility" viewpoint of k-center – i.e., we have a bipartite graph with facilities on one side and clients on the other (the LP can be naturally modified for this case, and it is known that it has an integrality gap at most 8 [21])[5]. Now on the facility side, we replicate a vertex $\min\{k, m_u\}$ times, and on the client side we have a *weight* of m_u for vertex u. The weight comes up only in the equation $\sum_u x_{uv} \leq L \cdot y_v$, which is replaced with $\sum_u m_u x_{uv} \leq L \cdot y_v$. Given a solution to the succint instance, we can find a feasible solution to the LP (it will not be integral – it will assign $x_{uv} = 1/m_u$ if one of the centers at location u is assigned to v). Note that the LP has N^2k variables, which is what gives the memory bound.

Now how do we round the LP? First, we will find the set of clusters to open – using the procedure of [21], which first clusters the facilities into clusters of small radius, forms a tree of clusters, and rounds by moving 'y'-mass between facilities in adjacent clusters. The key is to note that the rounding process depends only on the y_u (which exist only on the facility side), thus the rounding can be done with $O(Nk)$ space, after which we know the k facilities to open.

We now know that in the 'distance-8' graph, there is a feasible fractional solution to the LP in which we fix the values of y to be 0 or 1 based on the solution found in the rounding step above. So we can re-solve the LP (it still has at most N^2k variables). We claim that there exists an integer solution to the LP (we emphasize integer, not 0/1). This is done by viewing the LP as a solution to the following directed flow problem (source s and sink t). The edges in the middle (distance-8 graph) have capacity ∞, edges out of s

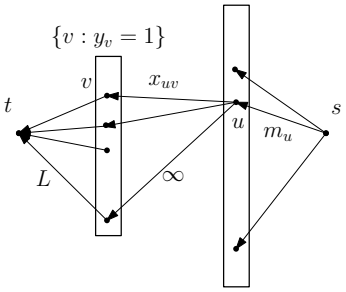

Figure 2: Flow network view of the LP.

have capacity equal to the multiplicity of the end point, and all the open facilities are connected to the sink with capacity L. Now we know that there is a solution of total flow $\sum_u m_u$. Thus there must be an integral flow of the same value (because all capacity values are integral).

The values of the flow can be used to read off the assignment. This completes the rounding.

6 Empirical study

In order to gauge its practicality, we implement our algorithm. We are interested in measuring its scalability in addition to the effect of having several rounds on the quality of the solution.

In particular, we compare the quality of the solution (i.e., the maximum radius from the k-center objective) produced by the parallel implementation to that of the sequential one-machine implementation of the farthest seed heuristic. In some sense, our algorithm is a parallel implementation of this algorithm. However, the instance is too big for the sequential algorithm to be feasible. As a result, we run the sequential algorithm on a small sample of the instance, hence a potentially easier instance.

[5]This variant is sometimes called the k-supplier problem.

Graph	Relative size of sequential instance	Relative increase in radius
US	0.33%	+52%
World	0.1%	+58%

Table 2: Quality degradation due to the two-round approach.

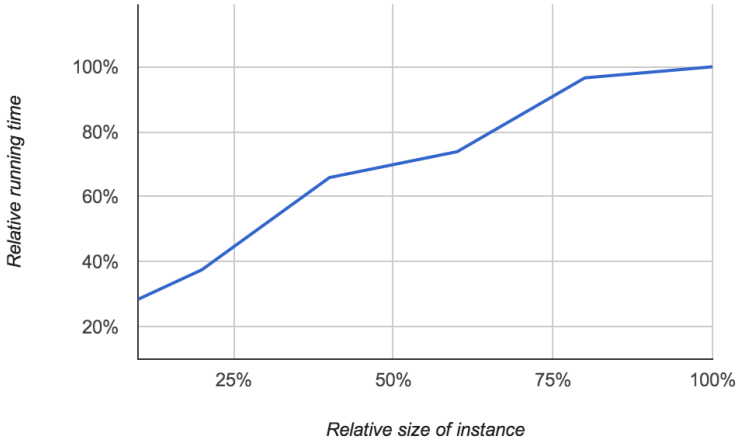

Figure 3: Scalability of parallel implementation.

Our experiments deal with two instances to test this effect: the larger instance is the world graph with hundreds of millions of nodes, and the smaller one is the graph of US road networks with tens of millions of nodes. Each node has the coordinate locations, which we use to compute great-circle distances—the closest distance between two points on the surface of the earth. We always look for 1000 clusters, and run our parallel algorithms on a few hundred machines.

Table 2 shows that the quality of the solution does not degrade substantially if we use the two-round algorithm, more suited to parallel implementation. The last column shows the increase in the maximum radius of clusters due to computing the k-centers in two rounds as described in the paper. Note that the radius increase numbers quoted in the table are upper bounds since the sequential algorithm could only be run on a simpler instance. In reality, the quality reduction may be even less. In case of the US Graph, the sequential algorithm was run on a random $\frac{1}{300}$ subset of the actual graph, whereas a random $\frac{1}{1000}$ subset was used for the World Graph.

We next investigate how the running time of our algorithm scales with the size of the instance. We focus on the bigger instance (World Graph) and once again take its random samples of different sizes (10% up to 100%). This yields to varying instance sizes, but does not change the structure of the problem significantly, and is perfect for measuring scalability. Figure 3 shows the increase in running time is sublinear. In particular, a ten-fold increase in instance size only leads to a factor 3.6 increase in running time.

References

[1] P. K. Agarwal, G. Cormode, Z. Huang, J. Phillips, Z. Wei, and K. Yi, *Mergeable summaries*, in Proceedings of the 31st symposium on Principles of Database Systems, ACM, 2012, pp. 23–34.

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

[3] H.-C. An, A. Bhaskara, and O. Svensson, *Centrality of trees for capacitated k-center*, CoRR, abs/1304.2983 (2013).

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

[5] M.-F. Balcan, S. Ehrlich, and Y. Liang, *Distributed clustering on graphs*, in NIPS, 2013, p. to appear.

[6] M. Bansal, N. Garg, and N. Gupta, *A 5-approximation for capacitated facility location*, in ESA, 2012, pp. 133–144.

[7] J. Bar-Ilan, G. Kortsarz, and D. Peleg, *How to allocate network centers*, J. Algorithms, 15 (1993), pp. 385–415.

[8] M. Charikar, C. Chekuri, T. Feder, and R. Motwani, *Incremental clustering and dynamic information retrieval*, in Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing, STOC '97, New York, NY, USA, 1997, ACM, pp. 626–635.

[9] M. Charikar, S. Guha, É. Tardos, and D. B. Shmoys, *A constant-factor approximation algorithm for the k-median problem*, J. Comput. Syst. Sci., 65 (2002), pp. 129–149.

[10] M. Charikar, L. O'Callaghan, and R. Panigrahy, *Better streaming algorithms for clustering problems*, in In Proc. of 35th ACM Symposium on Theory of Computing (STOC, 2003, pp. 30–39.

[11] J. Chuzhoy and Y. Rabani, *Approximating k-median with non-uniform capacities*, in SODA, 2005, pp. 952–958.

[12] M. Cygan, M. Hajiaghayi, and S. Khuller, *LP rounding for k-centers with non-uniform hard capacities*, in FOCS, 2012, pp. 273–282.

[13] J. Dean and S. Ghemawat, *Mapreduce: Simplified data processing on large clusters*, in OSDI, 2004, pp. 137–150.

[14] A. Ene, S. Im, and B. Moseley, *Fast clustering using mapreduce*, in KDD, 2011, pp. 681–689.

[15] D. Feldman and M. Langberg, *A unified framework for approximating and clustering data*, in Proceedings of the 43rd Annual ACM Symposium on Theory of Computing, STOC '11, New York, NY, USA, 2011, ACM, pp. 569–578.

[16] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan, *Clustering data streams*, STOC, (2001).

[17] A. Gupta and K. Tangwongsan, *Simpler analyses of local search algorithms for facility location*, CoRR, abs/0809.2554 (2008).

[18] P. Indyk, *Algorithms for dynamic geometric problems over data streams*, in Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing, STOC '04, New York, NY, USA, 2004, ACM, pp. 373–380.

[19] P. Indyk, S. Mahabadi, M. Mahdian, and V. Mirrokni, *Composable core-sets for diversity and coverage maximization*, in unpublished, 2014.

[20] H. J. Karloff, S. Suri, and S. Vassilvitskii, *A model of computation for mapreduce*, in SODA, 2010, pp. 938–948.

[21] S. Khuller and Y. J. Sussmann, *The capacitated k-center problem*, SIAM J. Discrete Math., 13 (2000), pp. 403–418.

[22] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, *Analysis of a local search heuristic for facility location problems*, in SODA, 1998, pp. 1–10.

[23] C. Lammersen and C. Sohler, *Facility location in dynamic geometric data streams*, in ESA, 2008, pp. 660–671.

[24] S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii, *Filtering: a method for solving graph problems in mapreduce*, in SPAA, 2011, pp. 85–94.

[25] J.-H. Lin and J. S. Vitter, *Approximation algorithms for geometric median problems*, Inf. Process. Lett., 44 (1992), pp. 245–249.

[26] F. Rahimian, A. H. Payberah, S. Girdzijauskas, M. Jelasity, and S. Haridi, *Ja-be-ja: A distributed algorithm for balanced graph partitioning*, in SASO, 2013, pp. 51–60.

[27] J. Ugander and L. Backstrom, *Balanced label propagation for partitioning massive graphs*, in WSDM, 2013, pp. 507–516.