Distributed Clustering via LSH Based Data Partitioning

Aditya Bhaskara [1] **Maheshakya Wijewardena** [1]

Abstract

Given the importance of clustering in the analysis of large scale data, distributed algorithms for formulations such as k-means, k-median, etc. have been extensively studied. A successful approach here has been the "reduce and merge" paradigm, in which each machine reduces its input size to $\widetilde{O}(k)$, and this data reduction continues (possibly iteratively) until all the data fits on one machine, at which point the problem is solved locally. This approach has the intrinsic bottleneck that each machine must solve a problem of size $\geq k$, and needs to communicate at least $\Omega(k)$ points to the other machines. We propose a novel data partitioning idea to overcome this bottleneck, and in effect, have different machines focus on "finding different clusters". Under the assumption that we know the optimum value of the objective up to a poly(n) factor (arbitrary polynomial), we establish worst-case approximation guarantees for our method. We see that our algorithm results in lower communication as well as a near-optimal number of 'rounds' of computation (in the popular MapReduce framework).

1. Introduction

Clustering is a fundamental problem in the analysis and understanding of data, and is used widely in different areas of science. The broad goal of clustering is to divide a (typically large) dataset into groups that such that data points within a group are "similar" to one another. In most applications, there is a measure of similarity between any two objects, which typically forms a metric. The problem can be formalized in many different ways, depending on the properties desired of the obtained clustering. While a "perfect" formulation may not exist (see (Kleinberg, 2002)), many formulations have been very successful in applications, including k-means, k-median, k-center, and various notions of hierarchical clustering (see (Hastie et al., 2009; Dasgupta, 2016) and references there-in).

In this paper, we focus on k-means clustering, in which the input is a set of n points in Euclidean space. Here the goal is to partition the points into k clusters, so as to minimize the sum of squared distances from the points to the respective cluster centers (see Section 2 for a formal definition). k-means is one of the most well-studied clustering variants. Lloyd's algorithm (Lloyd, 1982), developed over 35 years ago, has been extremely successful in practice (the success has been 'explained' in many recent works; see (Arthur et al., 2011; Awasthi & Sheffet, 2012; Kumar & Kannan, 2010) and references there-in). Despite the success, Lloyd's algorithm can have an arbitrarily bad approximation ratio in the worst case. To address this, constant factor approximation algorithms have been developed, which are more involved but have worst case guarantees (see (Kanungo et al., 2004) and (Ahmadian et al., 2017)). In another direction, works by (Ostrovsky et al., 2006; Jaiswal et al., 2012; Arthur & Vassilvitskii, 2007) have shown how to obtain simple bicriteria approximation algorithms for k-means. (Arthur & Vassilvitskii, 2007) also proposed a variant of Lloyd's algorithm, termed "k-means++", which also comes with a theoretical approximation factor guarantee of $O(\log k)$ approximation.

All the algorithms above assume that data fits in a single machine. However, with the ubiquity of large data sets, there has been a lot of interest in distributed algorithms where data is spread across several machines. The goal is to use available distributed models of computation to design algorithms that can (a) work with machines having access only to their local data set, (b) use small amount of memory and only a few "rounds" of communication, and (c) have approximation guarantees for the solution they output.

For k-means and related objectives, the paradigm of iterative 'data reduction' has been remarkably successful. The main idea is that in each round, a machine chooses a small subset of its input, and only this subset is carried to the next round. Thus the total number of points reduces by a significant factor in every round, and this results in a small number of rounds overall. Such an algorithm can be implemented

[*]Equal contribution [1]School of Computing, University of Utah. Correspondence to: Aditya Bhaskara <bhaskaraaditya@gmail.com>, Maheshakya Wijewardena <pmaheshakya4@gmail.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

efficiently in the MapReduce framework, introduced by (Dean & Ghemawat, 2004), and formalized by (Karloff et al., 2010)). (Ene et al., 2011) gave one of the first such implementations (for the k-median problem), and showed theoretical guarantees. This line of work has subsequently been developed in (Kumar et al., 2013; Balcan et al., 2013a; Awasthi et al., 2017). The last work also gives a summary of the known results in this space.

The high level ideas used in these works are similar to those used in streaming algorithms for clustering. The literature here is very rich; one of the earliest works is that of (Charikar et al., 1997), for the k-center problem. The work of (Guha et al., 2001) introduced many ideas crucial to the distributed algorithms mentioned above. Indeed, all of these algorithms can be viewed as implicitly constructing *coresets* (or summaries) for the underlying clustering problem. We refer to the works of (Agarwal et al., 2004; 2012; Balcan et al., 2013b; Indyk et al., 2014) for more on this connection.

Motivation for our work. While iterative data reduction is powerful, it has a key bottleneck: in order to have approximation guarantees, machines always need to store $> k$ data points. Indeed, all the algorithms we are aware of require a memory of kn^{ϵ} if they are to use $O(1/\epsilon)$ rounds of MAPREDUCE computation.[1] The high level reason for this is that if a machine sees k points that are all very far from one another, it needs to keep all of them, or else we might lose all the information about one of the clusters, and this could lead to a large objective value. This is also the reason each machine needs to communicate $\geq k$ points to the others (such a lower bound was proved formally in (Chen et al., 2016), as we will discuss later). The natural question is thus to ask:

can we partition the data across machines so that different machines work in different "regions of space", and thus focus on finding different clusters?

This would result in a smaller space requirement per machine, and lesser communication between machines. Our main result is to show that this is possible, as long as we have a *rough* estimate of the optimum objective value (up to an arbitrary polynomial factor). We give an algorithm based on a variant of locality sensitive hashing, and prove that this yields a bi-criteria approximation guarantee.

Locality sensitive hashing was introduced in the seminal work of (Indyk & Motwani, 1998), which gave an efficient algorithm for nearest neighbor search in high dimensional space. The idea has found several applications in machine learning and data science, ranging from the early applications of similarity search to the speeding up of neural networks (Spring & Shrivastava, 2017). (Datar et al., 2004) generalized the original result of (Indyk & Motwani, 1998) to the case of ℓ_p norms, and (Andoni & Indyk, 2006) gave an improved analysis. Extensions of LSH are still an active area of research, but a discussion is beyond the scope of this paper. Our contribution here is to understand the behavior of *clusters* of points under LSH and its variants.

[1] All the algorithms mentioned above can be naturally implemented in the MAPREDUCE framework.

1.1. Our results

Our focus in the paper will be on the k-means objective (defined formally in Section 2). The data set is assumed to be a collection of points in a Euclidean space $\mathbb{R}^d$ for some d, and distance refers to the ℓ_2 distance.

Our first contribution is an analysis of "product LSH" (PLSH), a hash obtained by concatenating independent copies of an LSH. For each LSH, we consider the implementation of (Andoni & Indyk, 2006).

Informal theorem 1 (See Lemmas 1 and 2)**.** *Let C be any cluster of points with diameter σ. Then PLSH with appropriate parameters yields the same hash for all the points in C, with probability $\geq 3/4$. Furthermore, for any two points u, v such that $\|u - v\| \geq \alpha \cdot \sigma$, where $\alpha \approx \log n \log\log n$, the probability that u and v have the same hash is $< 1/n^2$.*

Thus, PLSH has a "cluster preserving" property. We show the above by extending the analyses of (Indyk & Motwani, 1998) and (Andoni & Indyk, 2006). Then, we use this observation to give a simple bi-criteria approximation algorithm for k-means clustering. (A bi-criteria algorithm is one that is allowed to output a slightly larger number of centers; see Section 2.) We assume knowledge of k, as well as a *very rough* estimate of the objective value. The algorithm returns a polylogarithmically larger number of clusters, while obtaining a polylogarithmic factor approximation. We refer to Theorem 2 for the statement. As we note below, if $s > k$ polylog$(())n)$, then we can avoid violating the bound on the number of clusters (and obtain a "true" guarantee as opposed to a bi-criteria one).

The algorithm can be implemented in a distributed manner, specifically in the MAPREDUCE model, with $\lceil \log_s n \rceil + 2$ rounds, using machines of memory s (when we say memory s, we mean that each machine can store at most s of the points. This will be roughly the same as measuring s in bytes, as we see in Section 2. The formal result is stated in Theorem 3. We highlight that the distributed algorithm works for any $s \geq \omega(\log n)$, even $s \ll k$ (in which case the standard reduce-and-merge framework has no non-trivial guarantees).

Finally, we prove that for any MapReduce algorithm that uses poly(n) machines of space s, the number of rounds necessary to obtain *any non-trivial* approximation to k-means is at least $\lceil \log_s n \rceil$. Thus the 'round/memory tradeoff' we

obtain is nearly optimal. This is based on ideas from the recent remarkable result of (Roughgarden et al., 2016). (See Theorem 4.)

1.2. Discussion, extensions and limitations

Going beyond communication lower bounds. The recent result of (Chen et al., 2016) shows lower bounds on the total amount of communication necessary for distributed clustering. They show that for a worst-case partition of points across machines, $\Omega(Mk)$ bits are necessary, where M is the number of machines. Our result in Section 4.2 implies that if points have been partitioned across machines according to PLSH hashes, we can bypass this lower bound.

Round lower bound. In light of Theorem 4, one way to interpret our algorithmic result is as saying that as far as obtaining polylogarithmic bi-criteria approximations go, clustering is essentially as easy as "aggregation" (i.e., summing a collection of n numbers – which also has the same $\log_s n$ upper and lower bounds).

Precisely k clusters. Theorem 3 gives only a bi-criteria guarantee, so it is natural to ask if we can obtain any guarantee when we desire *precisely* k centers. In the case when $s \geq k \log^2 n$, we can apply known results to obtain this. (Guha et al., 2001) showed (in our notation) that:

Theorem 1. *Let U be a set of points, and let S be a set of centers with the property that $\sum_{u \in U} d(u, S)^2 \leq \gamma \cdot OPT$, where OPT is the optimum k-means objective value on U. Let $g : U \mapsto S$ map every $u \in U$ to its closest point in S, breaking ties arbitrarily. Now, consider a new weighted instance $\mathcal{I}$ of k-means where we have points in S, with weight of $v \in S$ being $|g^{-1}(v)|$. Then, any set of centers that ρ-approximate the optimum objective for $\mathcal{I}$ give a $(4\gamma + 2\rho)$ approximation to the original instance (given by U).*

Thus, if $s \geq k \log^2 n$, we can aggregate the output of our bi-criteria algorithm onto one machine, and solve k-means approximately. In essence, we are using the output of our algorithm as a *coreset* for k-means. We demonstrate this in our experimental results.

Balanced clustering. A common constraint for clustering algorithms is that of the clusters being *balanced*. This is often captured by requiring an upper bound on the size of a cluster. (Bateni et al., 2014) showed that balanced clustering can also be solved in a distributed setting. Specifically, they showed that any bi-criteria algorithm for k-means can be used to solve the balanced clustering problem, via a result analogous to 1. In our context, this implies that if $s > k \log^2 n$, our method also gives a distributed algorithm for balanced clustering with a k-means objective.

Limitations and lower bounds. There are two key limitations to our result. First, the polylogarithmic approximation factor in the approximation ratio seems difficult to avoid (although our experiments show that the guarantees are very pessimistic). In our argument, it arises as a by-product of being able to detect very small clusters. This is in contrast with single machine algorithms (e.g., (Kanungo et al., 2004; Ahmadian et al., 2017)) and the prior work in MapReduce algorithms, (Ene et al., 2011), which give constant factor approximations. Another restriction is that our algorithms assume a Euclidean setting for the points. The algorithms of (Ene et al., 2011) and related works can handle the case of arbitrary metric spaces. The bottleneck here is the lack of locality sensitive hashing for such spaces. A very interesting open problem is to develop new methods in this case, or prove stronger lower bounds.

2. Notation and Preliminaries

We now introduce some notation and definitions that will be used for the rest of the paper. We will denote by U the set of points in the input. We denote $n = |U|$. All of our algorithms are for the *Euclidean* setting, where the points in U are in $\mathbb{R}^d$, and the distance between $x, y \in U$ is the ℓ_2 norm $\|x - y\|_2 = \sqrt{\sum_i (x_i - y_i)^2}$.

A k-clustering of the points U is a partition $\mathcal{C}$ of U into subsets $C_1, C_2, \ldots, C_k$. The centroid of a cluster C_i is the point $\mu_i := \frac{1}{|C_i|} \sum_{u \in C_i} u$. The k-means objective for the clustering $\mathcal{C}$ is now defined as

$$\sum_{i \in [k]} \sum_{u \in C_i} \|u - \mu_i\|_2^2. \tag{1}$$

The problem of k-means clustering is to find $\mathcal{C}$ that minimizes the objective defined above. The minimum objective value will be denoted by $\mathrm{OPT}(U)$. (When the U is clear from context, we simply write OPT.) A ρ-approximation algorithm for k-means clustering is a polynomial time algorithm that outputs a clustering $\mathcal{C}'$ whose objective value is at most $\rho \cdot \mathrm{OPT}(U)$. We will be interested in ρ being a constant or $\mathrm{polylog}(n)$. A (ρ, β) bi-criteria approximation (where $\beta \geq 1$) is an efficient algorithm that outputs a clustering $\mathcal{C}'$ that has at most βk clusters and has an objective value at most $\rho \cdot \mathrm{OPT}(U)$. Note that the optimum still has k clusters.

Note on the dimension d. We assume that $d = O(\log n)$. This is without loss of generality, because we may assume that we have pre-processed the data by applying Johnson-Lindenstrauss transform. As the JL transform preserves all pairwise ℓ_2 distances (Johnson & Lindenstrauss, 1984; Indyk & Motwani, 1998), clustering in the transformed space gives a $(1 + \epsilon)$ approximation to clustering in the original one. Furthermore, the transform can be applied in parallel, individually to each point. Thus we henceforth assume that the space required to store a point is $O(\log n)$.

MapReduce model. To illustrate our ideas, we use the well-studied MapReduce model (Dean & Ghemawat, 2004; Karloff et al., 2010). The details of the map and reduce operations are not important for our purpose. We will view it as a model of computation that proceeds in *levels*. At each level, we have M machines that can perform computation on their local data (the input is distributed arbitrarily among machines in the first layer). Once all the machines are done with computation, they send information to machines in the next layer. The information received by a machine acts as its local data for the next round of computation. We assume that each machine has a *memory* of s.

Constants. For the sake of easier exposition, we do not attempt to optimize the constants in our bounds.

3. Two Step Hashing

The key to our distributed algorithm is a two step hashing scheme. The first step is a 'product' of locality sensitive hashes (LSH), and the second is a random hash that maps the tuples obtained from the product-LSH to a *bin* with a label in the range $1, \ldots, Lk$, for an appropriate constant L.

3.1. Product LSH

We begin with a short discussion of LSH. We follow the presentation of (Andoni & Indyk, 2006).[2]

Suppose we have a collection of points U in $\mathbb{R}^d$.

Locality sensitive hashing (LSH). Let t, w be parameters. $\text{LSH}_{t,w}$ is a procedure that takes a $u \in U$, and produces a $(t+1)$-tuple of integers. The hash uses as parameters a matrix A of dimensions $t \times d$, whose entries are i.i.d. $\mathcal{N}(0,1)$ Gaussians, and a collection of *shift vectors* $S = \{s_1, \ldots, s_U\}$, where s_i is picked uniformly from $[0, 4w]^t$. The shifts are used to generate a collection of shifted grids $G_i^t := G^t + s_i$, where G^t is the integer grid $\mathbb{Z}^t$, scaled by $4w$. Now to compute the hash of a point u, first its projection to $\mathbb{R}^t$ is computed by $u' = Au$. Next, one searches for the smallest index $i \in [U]$ for which the ball $B(u', w)$ contains a point of the shifted grid G_i^t. (Alternately one could imagine radius-w balls around the points in the shifted grids, and we look for the smallest i for which the point u' is contained in one such ball.) The hash of the point is then $(i, x_1, \ldots, x_t)$, where i is the index as above, and $(x_1, \ldots, x_t)$ are the integer coordinates corresponding to the grid point in G_i^t that is at distance $\leq w$ from u'.

(Andoni & Indyk, 2006) show that to cover all of $\mathbb{R}^t$ (and thus to have a well-defined hash for every point), the number of shifts that suffice is $2^{O(t \log t)}$. Consequently, this is also the time required to compute hash for a point, as we may need to go through all the shifts. In our setting, we will choose $t = o(\log n / \log\log n)$, and thus the time needed to hash is $n^{o(1)}$.

Product LSH (PLSH). Given an integer ℓ, the product LSH $\text{PLSH}_{t,w,\ell}$ is a hashing scheme that maps a point u to a concatenation of ℓ independent copies of $\text{LSH}_{t,w}$; it thus outputs an $\ell(t+1)$-tuple of integers.

We show the following properties of PLSH. In what follows, let σ be a parameter. Let

$$w = 8\sigma(\log n)^{3/2};\ t = \frac{\log n}{(\log\log n)^2};\ \ell = 32(\log\log n)^2. \tag{2}$$

Lemma 1. *Suppose $C \subseteq U$ has diameter $\leq \sigma$. Then with probability at least $3/4$, $\text{PLSH}_{t,w,\ell}$ maps all the points in C to the same hash value.*

Lemma 2. *Let u, v be points that are at distance $\geq 4w/\sqrt{t}$ $(= O(\log n \log\log n) \cdot \sigma)$. Then the probability that they have the same hash value is $< \frac{1}{n^4}$.*

Proof of Lemma 1. Let $C' = \{Ax : x \in C\}$. First, we claim that the diameter of C' is at most $4\sigma\sqrt{t + \log n}$, w.h.p. over the choice of A. This is because for any $x, y \in C$, the quantity $\|A(x-y)\|_2^2 / \|x-y\|_2^2$ is distributed as a χ^2 with t degrees of freedom. It is known (e.g., (Laurent & Massart, 2000), Lemma 1) that the tail of a chi-square statistic Y with t degrees of freedom satisfies: for any $z > 0$, $\Pr[Y \geq t + 2\sqrt{tz} + 2z] \leq e^{-z}$. Setting $z = 4\log n$, and using $2\sqrt{tz} \leq t + z$, we get that $\Pr[Y \geq 16(t + \log n)] < 1/n^4$. Thus by taking union bound over all pairs of points x, y, we have that with probability $\geq 1 - \frac{1}{n^2}$, the diameter of C' is $\leq 4\sigma\sqrt{t + \log n}$.

Conditioned on this, let us calculate the probability that the points in C all have the same hash value. (The conditioning does not introduce any dependencies, as the above argument depended only on the choice of A, while the next step will depend on the choice of the shifts.) Now, consider a ball $B^* \subseteq \mathbb{R}^t$ of radius $r' := 4\sigma\sqrt{t + \log n}$ that contains C' (as C' is of small diameter, such a ball exists).

Before analyzing the probability of interest, let us understand when a shifted grid G_i^t contains a point that has distance $\leq w$ to a given point x. This is equivalent to saying that $(x - s_i)$ is w-close to a lattice point in G^t. This happens iff s_i is in the ball $B(x, w)$, where the ball has been reduced modulo $[0, 4w]^t$ (see Figure 1).

Now, we can see how it could happen that some $x \in B^*$ is w-close to a lattice point in G_i^t but the entirety of B^* does not have this property. Geometrically, the bad choices of s_i are shown in Figure 1 (before reducing moulo $[0, 4w]^t$). Thus, we have that the probability that *all points in B^** are w-close to a lattice point in G_i^t conditioned on there existing

[2] The earlier LSH schemes of (Indyk & Motwani, 1998) and (Datar et al., 2004) can also be used; however, they give a weaker approximation factor.

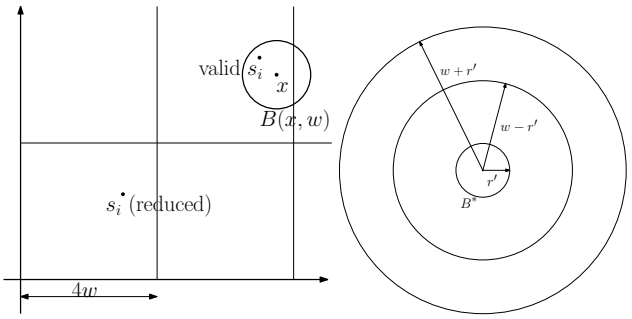

Figure 1. On the left, figure showing when grid G_i^t has a point w-close to x. On the right, choosing s_i in the region between the two outer circles results in B^* having points that hash differently.

a point $x \in B^*$ that is close to a lattice point is at least

$$p_1 := \frac{(w-r')^t}{(w+r')^t} = \left(1 - \frac{2r'}{w+r'}\right)^t.$$

Thus p_1 is a lower bound on a single LSH giving the same hash value for all the points in C. Repeating this ℓ times, and plugging in our choice of values for r', w, t, ℓ, the desired claim follows. □

Next, we get to the proof of Lemma 2.

Proof. Let u, v be points as in the statement of the lemma. We show that the probability that $\|A(u-v)\| \leq 2w$ in *all* the ℓ hashes is $< 1/n^4$. This clearly implies what we need, because if in even one LSH we have Au and Av being $> 2w$ away, they cannot have the same PLSH.[3]

Now, for a random A, the quantity $\|A(u-v)\|_2^2/\|u-v\|_2^2$ is distributed as a χ^2 distribution with t degrees of freedom (as we saw earlier), and thus using the *lower tail* from (Laurent & Massart, 2000), we get that for any $z > 0$, for such a random variable Y, we have $\Pr[Y \leq t - 2\sqrt{tz}] < e^{-z}$. Thus $\Pr[Y \leq (1-\frac{1}{\sqrt{2}})t] \leq e^{-t/8}$. Now, for our choice of parameters, we have $4w^2/\|u-v\|_2^2 \leq t/4 < (1-\frac{1}{\sqrt{2}})t$, and thus the probability that u and v have the same PLSH is upper bounded by $e^{-t\ell/8} = 1/n^4$, as desired. □

3.2. Number of tuples for a cluster

We have shown that the probability that a cluster of diameter $\leq \sigma$ hashes to precisely one tuple (for appropriate parameters) is $\geq 3/4$. We now show something slightly stronger (as we will need it later).

Lemma 3. *Let C be a cluster of diameter σ, and let t, w, ℓ be set as in Eq. (2). The expected number of distinct tuples for points in C (produced by $\text{PLSH}_{t,w,\ell}$) is $O(1)$.*

[3] This reasoning allows us to get a bound slightly better than (Andoni & Indyk, 2006).

3.3. Second step of hashing

The PLSH maps each point $u \in U$ to an $\ell(t+1)$ tuple of integers. The second step of hashing is very simple – we simply hash each tuple independently and uniformly to an integer in $[Lk]$, for a prescribed parameter L.

4. Approximation Algorithm

We start by describing our algorithm in a single machine setting. Then in Section 4.2, we describe how it can be implemented in parallel, with a small number of machines, and a small memory per machine.

4.1. Main algorithm

The high level idea in our algorithm is to perform the two-level hashing above, and choose a random subset of points from each bin.

Now, in order to choose the w parameter in the hash, we need a rough *scale* of the optimum. To this end, we will assume that we know a D such that the optimum objective value is in the range $(D/f, D)$, for some $f = \text{poly}(n)$. Note that f can be something like n^2, so this is a very mild assumption. With this assumption, we have that the average contribution of a point to the objective (i.e., its squared distance to its center) is $\geq D/(n \cdot f)$. Let us denote $r_0 := \sqrt{D/(n \cdot f)}$. Also, observe that no point can have a contribution more than D to the objective (as it is an upper bound on the sum of the contributions). Thus, inuitively, all the clusters have a radius (formally defined below) $\leq \sqrt{D}$. Let $\kappa = \lceil \log(nf) \rceil$, and let $r_i := 2^{i/2} r_0$, for $1 \leq i \leq \kappa$. These will be the different radius "scales" for our clusters of interest. Note that $\kappa = O(\log n)$, as $f = \text{poly}(n)$.

The algorithm can now be described (see Algorithm 4.1).

Algorithm 1 Find-Cover

Input: set of points U, rough estimate of optimum D.
Output: a subset of points S.
for $i = 1 \ldots \kappa$ **do**
- Hash every point in U to a bin (range $[Lk]$) using the two layer hash with params t, w_i, ℓ, Lk, where $w_i := 8r_i(\log n)^{3/2}$. Let U_j be the points hashed to bin j.
- Let G_j be the group of machines assigned for bin j. For each j, assign points in U_j uniformly at random to a machine in G_j.
- For each j, select a uniformly random subset of U_j of size $O(1)$ from G_j and add them to S. (If the number of points in the group is $O(1)$, add all of them to S.)

end for

In the remainder of this section, we analyze this algorithm. We start with a definition.

Definition 1 (Cluster radius)**.** *For a cluster C with centroid μ, we define the radius to be the quantity $\rho := \sqrt{\frac{1}{|C|}\sum_{p\in C}\|p-\mu\|_2^2}$, i.e., the "$\ell_2^2$ average" radius.*

Observation 1. *In any cluster C with centroid μ and radius ρ, the number of points p in C such that $\|p-\mu\|_2 > 2\rho$ is at most $|C|/4$.*

The proof follows by an averaging argument. Now, a candidate goal is to prove that for every optimum cluster C_i (center μ_i, radius ρ_i), the algorithm chooses at least one point at a distance $\leq \alpha\rho_i$ from the center μ_i with high probability, for some small α.

Unfortunately this statement is false in general. Suppose the instance has an optimal cluster with small ρ_i and a small $|C_i|$ that is really far from the rest of the points (thus it is essential to "find" that cluster). In this case, for r_j that is roughly ρ_i (which is the scale at which we hope to find a point close to this cluster), the bin containing C_i may contain many other points that are far away; thus a random sample is unlikely to choose any point close to C_i.

The fix for this problem comes from the observation that small clusters (i.e. small $|C_i|$) can afford to pay more per point to the objective. We thus define the notion of "adjusted radius" of a cluster. First, we define θ to be the real number satisfying $\text{OPT} = n\theta^2$, i.e., the typical distance of a point to its cluster center in the optimal clustering.[4] Now, we have:

Definition 2 (Adjusted radius)**.** *For a cluster C with radius ρ, we define the adjusted radius to be the quantity $\overline{\rho}$ satisfying $\overline{\rho}^2 = \rho^2 + \theta^2 + \frac{n\theta^2}{k|C|}$.*

Our main lemma about the algorithm is the following.

Lemma 4. *Let C be a cluster in the optimal clustering with adjusted radius $\overline{\rho}$. With probability $\geq 1/4$, Algorithm 4.1 outputs a point that is at a distance $\leq \alpha\cdot\overline{\rho}$ from the center of the cluster C, where $\alpha = O(\log n \log\log n)$.*

This is used to show the main result of this section.

Theorem 2. *Let S' be the union of the sets output by $O(\log k)$ independent runs of Algorithm 4.1. For $\alpha = O(\log n\log\log n)$, S' gives an $(\alpha^2, O(\log n\log k))$ bi-criteria approximation for k-means, w.p. at least $9/10$.*

Proof of Theorem 2 assuming Lemma 4. First, let us analyze the number of points output. Note that in each run of the algorithm, we output $O(Lk) = O(k)$ points for each radius range. There are $O(\log n)$ radius ranges and $O(\log k)$ independent runs. Thus we have the desired bound.

Next, consider the approximation factor. As we take S' to be the union of $O(\log k)$ independent runs of Algorithm 4.1, the success probability in Lemma 4 can be boosted to $1-\frac{1}{10k}$, and by a union bound, we have that the conclusion of the lemma holds for *all* clusters, with probability $> 1/10$. Thus for every optimal cluster C_i of adjusted radius $\overline{\rho}_i$, Algorithm 4.1 outputs at least one point at a distance $\leq \alpha\cdot\overline{\rho}_i$, for α as desired. Thus, assigning all the points in C_i to one such point would imply that the points of C_i contribute at most $|C_i|\rho_i^2 + \alpha^2|C_i|\overline{\rho}_i^2$ to the objective.[5] Thus the objective value is at most

$$\sum_i |C_i|\rho_i^2 + \alpha^2|C_i|\left(\rho_i^2 + \theta^2 + \frac{n\theta^2}{k|C_i|}\right)$$
$$= (1+\alpha^2)\text{OPT} + \alpha^2\cdot 2n\theta^2 \leq 4\alpha^2\text{OPT}.$$

This completes the proof. □

4.2. Distributed implementation

We now see how to implement algorithm from Theorem 2 in a distributed setting with a small number of rounds and machines, while also using memory $\leq s$ per machine. Our final result is the following.

Theorem 3. *There is a distributed algorithm that performs $\lceil\log_s n\rceil + 2$ rounds of* MapReduce *computation, and outputs a bi-criteria approximation to k-means, with the same guarantee as Theorem 2. The number of machines needed is $\widetilde{O}\left(\frac{n}{\min\{k,s\}}\cdot\frac{k}{s}\right)$, and the space per machine is s.*

Note. Whenever $s \geq k$, the bound on the number of machines is $\widetilde{O}(n/s)$, which is essentially optimal, because we need n/s machines to hold n points, if each machine has a memory of s.

While most parts of the algorithm from Theorem 2 can be immediately parallelized, sampling from U_j (which may need to be split across machines) is the tricky part and requires some work. The proof is deferred to Section 3 of the supplement.

5. Lower Bound

We now show that even in the very simple setting of points on a line, we have a tradeoff between the number of rounds and memory. This matches the behavior of our algorithm, up to an additive constant.

Theorem 4. *Let α be any parameter that is poly(n). Then, any α factor approximation algorithm for k-means with $k \geq 2$ that uses poly(n) machines of memory $\leq s$ requires at least $\log_s n$ rounds of* MapReduce.

The proof is by a simple reduction from Boolean-OR,[6] a

[4] We note that θ is used solely for analysis purposes – the algorithm is not assumed to know it.

[5] This follows from a "center-of-mass" theorem that is standard: for a set T of points with centroid μ, and any other point μ', $\sum_{u\in T}\|u-\mu'\|^2 = \sum_{u\in T}\|u-\mu\|^2 + |T|\|\mu-\mu'\|^2$.

[6] The input is the set of bits $x_1,\ldots,x_n$, and the desired output

problem for which a round-memory trade-off was established in (Roughgarden et al., 2016).

Proof. Suppose we have inputs $x_1, \ldots, x_n$, the inputs for Boolean OR. We produce an instance of clustering with $k+n$ points, all on the line.

First, we place points at $1, 2, \ldots, k$. Additionally, for $1 \leq i \leq n$, if $x_i = 1$, we add a point at $k+\alpha+1$. If $x_i = 0$, add a point at 1. Now if the OR of the x_i's is TRUE, then the optimum solution places centers at $1, 2, \ldots, k-1, k+\alpha+1$. This results in an objective value of 1. If the OR is FALSE, the optimum solution is to place centers at $1, 2, \ldots, k$ (0 cost). Thus an α factor approximation should be able to distinguish between the two cases (because in the NO case, it needs to have error 0, and in the YES case, this solution will be a factor $> \alpha$ off. □

Note. The parameter α implicitly comes up in the reduction. The number of bits necessary to write down the points x_i is $n \log \alpha$. This is why we insist on $\alpha = \text{poly}(n)$.

The lower bound above can be extended in order to rule out both (a) the case in which we have a rough estimate of the optimum (as in our algorithm), and (b) bi-criteria approximations. To handle (a), we can perturb the NO case so as to make the objective $1/p(n\alpha)$ for a large polynomial $p(\cdot)$. In order to handle (b), i.e., to rule out an (α, β) bi-criteria approximation, we need to add a multiplicity of βk for each of the points in the proof above. This leads to a slightly weaker lower bound of $\log_s \frac{n}{k\beta}$ rounds. The details of these steps are straightforward, so we omit them.

6. Empirical Study

We evaluate our algorithmic ideas on two synthetic and two real datasets, of varying sizes. In the former case, we know the ground truth clustering, the "right k", and the optimum objective value. We use it to demonstrate how the quality of clustering depends on the parameter ℓ – the number of independent hashes we concatenate. In all the datasets, we compare the objective value obtained by our algorithm with the one obtained by k-means++ (part of scikit-learn (Pedregosa et al., 2011)). This will only be possible for small enough datasets, as k-means++ is a single machine algorithm that uses $\Omega(nk)$ memory.

6.1. Synthetic datasets

Both the datasets we consider are mixtures of Gaussians. The first has $n = 10^5$ points in $\mathbb{R}^{50}$ and $k = 100$, while the second has $n = 10^6$ point in $\mathbb{R}^{50}$ and $k = 1000$. For $i \in [k]$, the centers are chosen uniformly from a box of length 400 in each dimension, maintaining a distance of 20 from one another. To form cluster C_i, a random number of points are chosen from the Gaussian $\mathcal{N}(\mu_i, 1)$.

is simply the OR of the bits.

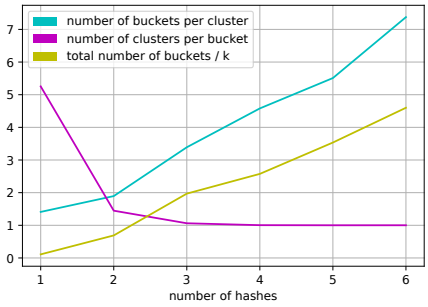

Figure 2. This figure shows how number of buckets per cluster, number of clusters per bucket, and the total number of buckets change as number of hashes increases.

For the first dataset, we produce PLSH tuples using $w = 15, t = 2$, and vary ℓ.We call the set of points that hash to a given tuple a *bucket* of points. We measure the following quantities: (a) the total number of non-empty buckets, (b) the "purity" of the buckets, i.e., the number of distinct clusters at intersect a non-empty bucket (on average), and (c) the "spread" of a cluster, i.e., the number of buckets that points of a cluster go to. The plots for these quantities as ℓ varies are shown in Figure 2. Note that it makes sense for the spread to increase as ℓ increases, as even a difference in one of the ℓ independent hashes results in unequal hashes.

Next, we study the objective value. For this we choose $\ell = 3$. This results in 257 non-empty buckets. Now, from each bucket, we output j points uniformly at random to form a set S (and do this for different j). Even for $j = 1$, the objective value is 41079, which is less than a factor 2 away from the optimum, 26820. This is significantly better than the guarantee we get from Theorem 2. It is also significantly better than a *random* subset of 257 points, for which it turns out that the objective is 5897317.

Intuitively, a random sample will be bad for this instance, because there are many clusters of size $\ll n/k$, and no points from such clusters will be chosen. This motivates us to measure the *cluster recall* of our procedure – how many clusters contribute to the 257 size set we output? Interestingly, *all* 100 of the clusters do, for the above values of the parameters. These results are consistent with the theoretical observations that *PLSH finds small-cardinality clusters while a random sample does not*.

Next, consider the larger synthetic dataset. Modulo n, k, the data is generated as before. Here, we produce PLSH tuples using $w = 15, t = 3$, and $\ell = 4$. For these choices of n and k, the single-machine k-means++ runs out of memory. However, as we know the μ_i, we can estimate the optimum

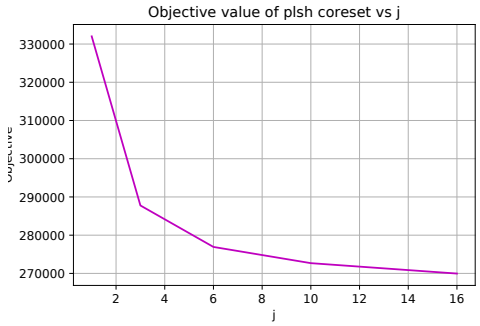

Figure 3. Objective value vs size of coreset (determined by j, the number of points sampled from each PLSH bucket).

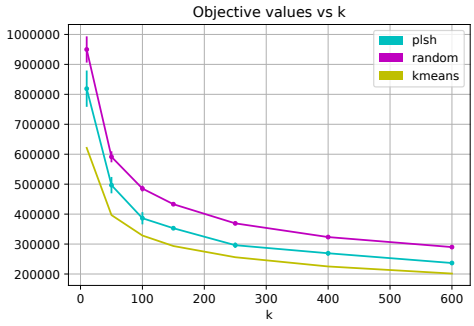

Figure 4. SUSY dataset: objective values of k-means++, PLSH, and a random set of points against k

objective value, which is 251208.

In this dataset, we illustrate the use of our algorithm to generate a *coreset*, as discussed in Section 1.2. We obtain 5722 buckets, from each of which we sample j points to obtain a coreset S. We then run k-means on S with $k = 1000$, thus obtaining 1000 centers. We evaluate the k-means objective with these centers. Results for different j are shown in Figure 3. Note that even with $j = 10$, the objective is within a 1.1 factor of the optimum.

6.2. Real datasets

We show our results on two datasets, both available via the UC Irvine dataset repository.

SUSY. The first dataset is SUSY (see (P. Baldi, 2014)), which contains 5M points with 18 features. In order to efficiently compare with k-means++, we use a sub-sample of 100000 points. In this case we use the values of t, ℓ as in our theoretical results. We also try different values for w. We start with a guess of $w = \sigma(\log n)^{3/2}$, where σ was obtained from k-means++ with $k = 10$ (which is very fast). We then scale σ from 2^{-4} to 2^2 in order to perform the hashing in different radius ranges. After hashing and finding S, we use it as a coreset and compute k-means. Figure 4 shows the results, and also compares against a fully random subset of points. Unlike the synthetic examples, here a random set of points is not orders of magnitude worse, but is still considerably worse than the output of our algorithm. We also note that our values are within a factor 1.2 of k-means++ (which is sequential and significantly slower). The number of buckets per cluster when $k = 600$ for $\ell = 1, \ldots, 6$ are $0.03, 0.31, 1.21, 3.97, 7, 10.55$.

FMA: A Dataset For Music Analysis This dataset (see (Defferrard et al., 2017)) contains 518 features extracted from audio files available in the free music archive (FMA). It has 106574 points. We perform the same experiment we did for the SUSY dataset. Figure 5 shows

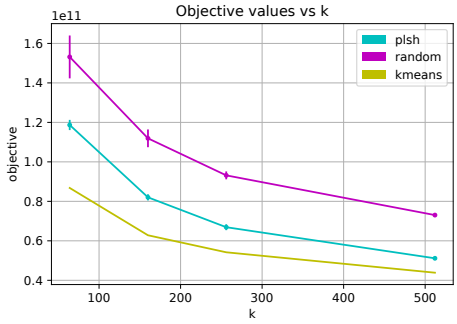

Figure 5. Music dataset: objective values of k-means++, PLSH, and a random set of points against k

the results, comparing the outputs with the output of k-means++, as well as a random subset. The number of buckets per cluster when $k = 512$ for $\ell = 1, \ldots, 6$ are $0.08, 0.68, 2.29, 5.27, 9.43, 14.09$ respectively.

References

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

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

Ahmadian, S., Norouzi-Fard, A., Svensson, O., and Ward, J. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 61–72, 2017.

Andoni, A. and Indyk, P. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In

47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings, pp. 459–468, 2006. doi: 10.1109/FOCS.2006.49.

Arthur, D. and Vassilvitskii, S. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pp. 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 978-0-898716-24-5.

Arthur, D., Manthey, B., and Röglin, H. Smoothed analysis of the k-means method. *J. ACM*, 58(5):19:1–19:31, October 2011. ISSN 0004-5411. doi: 10.1145/2027216.2027217.

Awasthi, P. and Sheffet, O. Improved spectral-norm bounds for clustering. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, pp. 37–49, 2012. doi: 10.1007/978-3-642-32512-0_4.

Awasthi, P., Balcan, M., and White, C. General and robust communication-efficient algorithms for distributed clustering. *CoRR*, abs/1703.00830, 2017.

Balcan, M., Ehrlich, S., and Liang, Y. Distributed k-means and k-median clustering on general communication topologies. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pp. 1995–2003, 2013a.

Balcan, M.-F., Ehrlich, S., and Liang, Y. Distributed clustering on graphs. In *NIPS*, pp. to appear, 2013b.

Bateni, M., Bhaskara, A., Lattanzi, S., and Mirrokni, V. S. Distributed balanced clustering via mapping coresets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 2591–2599, 2014.

Charikar, M., Chekuri, C., Feder, T., and Motwani, R. Incremental clustering and dynamic information retrieval. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pp. 626–635, New York, NY, USA, 1997. ACM. ISBN 0-89791-888-6. doi: 10.1145/258533.258657.

Chen, J., Sun, H., Woodruff, D. P., and Zhang, Q. Communication-optimal distributed clustering. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 3720–3728, 2016.

Dasgupta, S. A cost function for similarity-based hierarchical clustering. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pp. 118–127, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4132-5. doi: 10.1145/2897518.2897527.

Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, pp. 253–262, New York, NY, USA, 2004. ACM. ISBN 1-58113-885-7. doi: 10.1145/997817.997857.

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

Defferrard, M., Benzi, K., Vandergheynst, P., and Bresson, X. Fma: A dataset for music analysis. 2017.

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

Guha, S., Mishra, N., Motwani, R., and O'Callaghan, L. Clustering data streams. *STOC*, 2001.

Hastie, T., Tibshirani, R., and Friedman, J. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition, 2009.

Indyk, P. and Motwani, R. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pp. 604–613, New York, NY, USA, 1998. ACM. ISBN 0-89791-962-9. doi: 10.1145/276698.276876.

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

Jaiswal, R., Kumar, A., and Sen, S. A simple dˆ2-sampling based PTAS for k-means and other clustering problems. *CoRR*, abs/1201.4206, 2012.

Johnson, W. B. and Lindenstrauss, J. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in modern analysis and probability (New Haven, Conn., 1982)*, volume 26 of *Contemp. Math.*, pp. 189–206. Amer. Math. Soc., Providence, RI, 1984.

Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., and Wu, A. Y. A local search approximation algorithm for k-means clustering. *Computational Geometry*, 28(2):89 – 112, 2004. ISSN 0925-7721. doi:

https://doi.org/10.1016/j.comgeo.2004.03.003. Special Issue on the 18th Annual Symposium on Computational Geometry - SoCG2002.

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

Kleinberg, J. An impossibility theorem for clustering. In *Proceedings of the 15th International Conference on Neural Information Processing Systems*, NIPS'02, pp. 463–470, Cambridge, MA, USA, 2002. MIT Press.

Kumar, A. and Kannan, R. Clustering with spectral norm and the k-means algorithm. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pp. 299–308, 2010. doi: 10.1109/FOCS.2010.35.

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

Laurent, B. and Massart, P. Adaptive estimation of a quadratic functional by model selection. *Ann. Statist.*, 28 (5):1302–1338, 10 2000. doi: 10.1214/aos/1015957395.

Lloyd, S. P. Least squares quantization in pcm. *IEEE Trans. Information Theory*, 28:129–136, 1982.

Ostrovsky, R., Rabani, Y., Schulman, L. J., and Swamy, C. The effectiveness of lloyd-type methods for the k-means problem. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pp. 165–176, 2006. doi: 10.1109/FOCS.2006.75.

P. Baldi, P. Sadowski, D. W. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 2014. doi: 10.1038/ncomms5308.

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

Roughgarden, T., Vassilvitskii, S., and Wang, J. R. Shuffles and circuits: (on lower bounds for modern parallel computation). In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '16, pp. 1–12, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4210-0. doi: 10.1145/2935764.2935799.

Spring, R. and Shrivastava, A. Scalable and sustainable deep learning via randomized hashing. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pp. 445–454, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4887-4. doi: 10.1145/3097983.3098035.