# Fast and Accurate $k$-means++ via Rejection Sampling

**Vincent Cohen-addad**[*]
Google Research
cohenaddad@google.com

**Silvio Lattanzi**[*]
Google Research
silviol@google.com

**Ashkan Norouzi-Fard**[*]
Google Research
ashkannorouzi@google.com

**Christian Sohler**[*†]
University of Cologne
csohler@uni-koeln.de

**Ola Svensson**[*]
EPFL
ola.svensson@epfl.ch

## Abstract

$k$-means++ [4] is a widely used clustering algorithm that is easy to implement, has nice theoretical guarantees and strong empirical performance. Despite its wide adoption, $k$-means++ sometimes suffers from being slow on large data-sets so a natural question has been to obtain more efficient algorithms with similar guarantees. In this paper, we present a near linear time algorithm for $k$-means++ seeding. Interestingly our algorithm obtains the same theoretical guarantees as $k$-means++ and significantly improves earlier results on fast $k$-means++ seeding. Moreover, we show empirically that our algorithm is significantly faster than $k$-means++ and obtains solutions of equivalent quality.

## 1   Introduction

Clustering is a fundamental task in machine learning with many applications in data analysis and machine learning. One particularly important variant is $k$-means clustering: Given a set of $n$ points in $\mathbb{R}^d$ the goal is to find a partition of the points into $k$ clusters such that the sum of squared distance to the cluster centers (which are the means of the clusters) is minimized.

A popular method to compute a good clustering with respect to the $k$-means objective is the K-MEANS++ algorithm [4]. The algorithm is a combination of a randomized procedure for finding a set of $k$ starting centers (often referred to as the *seeding*) with the classic local improvement algorithm by Lloyd [25]. The seeding step samples the first center uniformly at random. In the remaining iterations the algorithm samples a center from the $D^2$-distribution, where the probability of sampling a point is proportional to the squared distance to the current set of centers.

The K-MEANS++ algorithm is easy to implement, has strong theoretical guarantees (an $O(\log k)$ approximation guarantee), and performs empirically well. However, the running time of $\Theta(dnk)^3$ becomes impractical for many very large datasets. Therefore, a lot of previous work focused on speeding up the K-MEANS++ seeding [5, 6] as well as Lloyd's algorithm [18, 12, 27, 15].

To obtain a fast seeding algorithm, Bachem et al. [5] and [6] use an MCMC algorithm to generate a set of $k$ centers that follows the K-MEANS++ distribution. They provide different versions of their algorithm that provide trade-offs between theoretical guarantees and empirical running time. Interestingly, under certain assumptions on the inputs and when $k$ is small, their algorithm may even run in sublinear time in the input size. However, all versions of their algorithms have a

---

[*]Equal contribution
[†]Work was partially done while author was visiting researcher at Google Research, Switzerland.
[3]Even assuming a constant number of Lloyd's algorithm steps.

running time of $\Omega(k^2)$[4] and so it does not scale for massive datasets and moderate values of $k$ (i.e. $500 - 1000$). Another important drawback from their results is that the guarantees on the quality of the solution output by their algorithms are weaker than the original K-MEANS++ guarantee (since their approximation is additive in the worst case).

**Our contribution** In this paper we present a new algorithm that provably achieves near-linear running time while offering similar guarantees as the original K-MEANS++ algorithm. In particular:

- We introduce a new seeding algorithm that for constant $\varepsilon > 0$ has near-linear running time $\widetilde{O}\left(nd + (n\log(\Delta))^{1+\varepsilon}\right)$ and returns a $O_\varepsilon(\log k)$ approximate solution, where $n$ is the number of points in the dataset and $\Delta$ is the ratio between the maximum distance and the minimum distance between two points in the dataset, see Corollary 5.5. Our algorithm also has the advantage that, in the stated running time, it computes the solution for *all* values of $k = 1, 2, \ldots, n$.
- We compare the performances of our seeding technique with the baselines K-MEANS++ and AFKMC2 from [5] on various classic datasets. Our algorithms outperform the baselines even for moderate values of $k$ (e.g.: $k = 1000$) and the speed-up is an order of magnitude for larger values of $k$[5]. In addition, our algorithms output solutions of similar costs as K-MEANS++ (as our theoretical results predict).

The main idea behind our method is to use an embedding into a collection of trees to approximate the distances between the input points, and then leverage the tree structure to speed-up the $D^2$-sampling of K-MEANS++. To ensure that our sampling, which uses the approximate tree distances, leads to a solution that is competitive with respect to K-MEANS++ on the *original* data, we "emulate" the $D^2$-distribution on the original data by additionally using rejection sampling. More concretely,

– We first develop a new seeding algorithm FASTK-MEANS++ that computes in $\widetilde{O}(nd)$ time a solution. The near linear running time is obtained by first approximating the squared Euclidean distance using a multi-tree embedding and then by showing that one can efficiently perform $D^2$-sampling with respect to multi-tree distances.

– We then argue that one can use our sampling technique on multi-tree distances in combination with rejection sampling so as to reproduce the same distribution as used by K-MEANS++ on the original distances and so to achieve the same approximation guarantees that K-MEANS++. To ensure a fast running time, we calculate the rejection probability by using locality-sensitive hashing (LSH) to approximately determine the nearest neighbor w.r.t. the original distances.

– Finally, we show that our LSH based rejection sampling algorithm computes a solution with the same expected approximation guarantee of $O(\log k)$ as the basic K-MEANS++ algorithm.

## 2 Preliminaries

**Basic notation.** We denote by $P \subseteq \mathbb{R}^d$ the set of $n$ input points in a $d$ dimensional space and let $\Delta$ be the ratio between the maximum distance and the minimum distance between two point in the dataset. The Euclidean distance between two points $x, y \in \mathbb{R}^d$ is denoted by $\text{DIST}(x, y) = ||x - y||_2$. We also let $\text{DIST}(x, C) = \min_{y \in C} \text{DIST}(x, y)$ be the distance of $x$ to the closest point in $C$. The goal in the $k$-means problem is to choose a set of $k$ centers $C \subseteq \mathbb{R}^d$ minimizing $\sum_{x \in P} \text{DIST}(x, C)^2$.

**Tree embeddings.** Tree embedding is a well-known technique used in many different clustering problems (see for example [7]). We now explain a simple version that is similar to [22] that will be used in our algorithm. We first compute an upper bound MAXDIST on the maximum distance between two points within a factor of 2.[6] Second, we add a random shift $0 \le s \le$ MAXDIST to each coordinate of all input points[7]. Let $x \in P$ be any point in the data set. The root of the tree (at

---

[4]We note that a similar running time can be achieved also via coresets [20, 13] but it is challenging to go below the $\Omega(k^2)$ barrier.

[5]While the large $k$ setting is not the most studied setting, it still has many practical applications. For instance in spam and abuse [28, 30], near-duplicate detection [21], compression or reconciliation tasks [29]. Furthermore the large $k$ case is very interesting from a theoretical perspective and it gained attention in recent years [10].

[6]This can be done in time $O(nd)$, by selecting any point and by computing the maximum distance between that point and any other point in the dataset. Then multiply this distance by 2.

[7]Notice that this does not effect the distance between any two points and therefore the cost of any solution.

height zero) represents an axis-aligned cube of side length $2\text{MAXDIST}$ centered at $x$. By selection, note that all the input points are inside this cube and we say that they *belong to* this node of the tree. We then partition this cube into $2^d$ axis-aligned subcubes of side length $\text{MAXDIST}$ and assign each point to the one that contains its coordinates. For each of these subcubes that contains a point, we create a node and add it as a child of the root in the tree (so their height is one), with edge weight $\sqrt{d}\text{MAXDIST}$, i.e., the side length of the (parent) cube times $\sqrt{d}/2$. Notice that this is equal to the half of the maximum distance between any two coordinates in the parent cube. Also observe that the number of nodes at height one is at most $n$, since each node contains at least one point. We let the height of these edges be zero. We then repeat this operation on the nodes until *every* cube contains at most a single point. This results in a tree where all leaves are at the same height, the height is at most $H = O(\log(d\Delta))$, and there are at most $n$ nodes in each layer. Moreover, the running time of constructing each layer is $O(nd)$ since for each point we can determine in which subcube it belongs by going over its dimensions. The total running time is thus $O(nd\log(d\Delta))$. The distance between two points $p, q$ in the tree, denoted by $\text{TREEDIST}(p, q)$, is the length of the shortest path between $p$ and $q$ in the tree, or equivalently twice the length from one of them to their lowest common ancestor.

## 3 Multi-tree Embedding

Tree embedding is a powerful tool for designing approximation algorithms but it cannot be applied directly to the $k$-means problem. In fact there are simple examples that show that the expected distortion between the squared distances of an $\ell_2$ metric and the $\text{TREEDIST}$ is $\Omega(n)$. To overcome this limitation, we use *three* tree embeddings with different random shifts and we define the distance between two points (denoted by $\text{MULTITREEDIST}$) to be the minimum $\text{TREEDIST}$ among the distances in the three trees. Interestingly, we show that this suffices to get a significantly better upper bound on the distortion. We refer to this simple procedure as $\text{MULTITREEINIT}()$. We note that the running time of $\text{MULTITREEINIT}()$ is asymptotically equal to that of a single tree embedding $O(nd\log(d\Delta))$ since it initializes three tree embeddings.

To analyze the expected distortion, define for any set $S \subseteq P$ and point $p \in P$, $\text{MULTITREEDIST}(p, S) = \min_{q \in S} \text{MULTITREEDIST}(p, q)$. The proof of the following bounds is provided in Appendix A.

**Lemma 3.1** *For any point $p$, and set $S$, we have* $\text{DIST}(p, S)^2 \leq \text{MULTITREEDIST}(p, S)^2$ *and* $\mathbf{E}[\text{MULTITREEDIST}(p, S)^2] \leq O(d^2 \cdot \text{DIST}(p, S)^2)$.

## 4 FASTK-MEANS++ Algorithm

Recall that the classic K-MEANS++ algorithm samples the first center uniformly at random and in the remaining iterations $k$-means++ samples a center from the $D^2$-distribution, where the probability of each point is proportional to the squared distance to its nearest current center. The most expensive operation in this procedure is to update the $D^2$-distribution after each sample. In fact, the probability for a point to be selected may change in every round of the algorithm leading to $n$ updates in each of the $k$ iterations.

Our key idea here is to use the special structure of the the multi-tree embedding to update the $D^2$-distribution with respect to those distances efficiently. This is intuitively possible since in the multi-tree metric every node can change its distance from the current set of centers at most $O(\log(d\Delta))$ times. This is true because in order to decrease the distance between a point $x$ and the set of centers in a single tree embedding, the lowest common ancestor between $x$ and the closest center has to get closer to $x$. The number of times that this can happen is bounded by the height of the tree. Therefore, since the multi-tree embedding consists of three trees of height $O(\log(d\Delta))$, we have that the number of times a point can change its multi-tree distance to the set of opened centers is at most $O(\log(d\Delta))$.

**MULTITREEOPEN and MULTITREESAMPLE.** To describe our algorithm we start by defining the procedures to update the distribution, MULTITREEOPEN, and to compute a sample MULTI-TREESAMPLE. To achieve an efficient running time, both procedures act on a common data structure which consists of the following:

- A weight $w_x$ for each point $x \in P$.

- A node-weighted balanced binary tree with a leaf for each of the $n$ points in $P$. We refer to this tree as the sample-tree so as to not confuse it with the trees in the multi-tree embedding.

- For each node in each of the trees in the multi-tree embedding, a bit saying whether this node is marked.

For notational convenience, let $\textsc{MultiTreeDist}(x, \emptyset)^2 = M$ for any point $x \in P$, where $M = 16d \cdot \textsc{MaxDist}^2$ is chosen to be an upper bound of $\textsc{MultiTreeDist}(p, q)^2$ for any two points $p$ and $q$. If we let $S$ be the set of opened points (using calls to $\textsc{MultiTreeOpen}$), the data structure will satisfy the following invariants:

1. For every $x \in P$, $w_x = \textsc{MultiTreeDist}(x, S)^2$.

2. The weight of each node in the sample-tree equals the sum of the weights of the points corresponding to the leaves in its subtree.

3. A node $v$ in a tree $T$ in the multi-tree embedding is marked if there is a point in its subtree that has been opened, i.e., is in $S$; otherwise it is in unmarked.

So the data structure is initialized (when $S = \emptyset$) by setting all weights $(w_x)_{x \in P}$ to $M$; setting the weight of each node in the sample-tree to $M$ times the number of points in its subtree; and by letting all nodes in the trees of the multi-tree embedding to be unmarked. In addition, for each tree $T$ in the multi-tree embedding and for each node $v$ in $T$, we compute the set $P_T(v) \subseteq P$ of points in its subtree. Note that the initialization of the weights and the sample-tree run in time $O(n)$ whereas the initialization of the unmarked notes and the sets $P_T(v)$ can be computed in time $O(n \log(d\Delta))$ by traversing the trees in the multi-tree embedding of height $O(\log(d\Delta))$. The total runtime of the initialization is thus $O(n \log(d\Delta))$.

We proceed to describe the procedure $\textsc{MultiTreeOpen}$ that opens a new point $x$ and updates the data structure to satisfy the invariants. We then describe the simpler procedure $\textsc{MultiTreeSample}$ which samples a point $x$ with probability $w_x / (\sum_{y \in P} w(y))$, i.e., from the $D^2$-distribution with respect to the multi-tree distances.

The description of $\textsc{MultiTreeOpen}$ is given in Algorithm 1. When the tree embedding is not clear from the context, we use the notation $\textsc{TreeDist}_T$ to denote the distances given by the tree embedding $T$. We now verify the invariants and give some intuition of the procedure. Let $S$ be the set of opened centers prior to this call to $\textsc{MultiTreeOpen}(x)$ and let $T$ be a tree in the multi-tree embedding. When considering $T$, $\textsc{MultiTreeOpen}$ starts in the leaf $v_0$ of $T$ that $x$ belongs to. It then traverses the tree towards the root forming a path $v_0, v_1, \ldots, v_\ell$ of nodes so that $v_\ell$ is either the root or its parent is already marked. The subtrees of these vertices are exactly those that contain $x$ but no other point in $S$, and so Step 4 guarantees the third invariant. Now a key observation is that $\textsc{TreeDist}_T(y, S \cup \{x\}) < \textsc{TreeDist}_T(y, S)$ for exactly those points $y$ in $P_T(v_\ell)$. This holds because in order to decrease the distance be-

---

**Algorithm 1** $\textsc{MultiTreeOpen}$

**Input:** A point $x \in P$
1: **for** each tree $T$ in the multi-tree embedding **do**
2:     Let $v_0$ be the leaf of $T$ that $x$ belongs to.
3:     Traverse the tree towards the root forming a path $v_0, v_1, \ldots, v_\ell$ until either $v_\ell$ is the root or the parent of $v_\ell$ is marked.
4:     Mark $v_0, \ldots, v_\ell$.
5:     **for** each point $y$ in $P_T(v_\ell)$ **do**
6:         **if** $\textsc{TreeDist}_T(y, x)^2 < w_y$ **then**
7:             $w_y \leftarrow \textsc{TreeDist}_T^2(y, x)$
8:             Traverse the sample-tree from the leaf corresponding to $y$ to the root to update the node-weights that depend on $w_y$.
9:         **end if**
10:     **end for**
11: **end for**

---

tween a point $y$ and the set of centers, with respect to the tree embedding $T$, the lowest common ancestor in $T$ between y and the closest center must get closer. $\textsc{MultiTreeOpen}$ considers each of these points and updates $w_y$ if $\textsc{TreeDist}_T(y, x) < w_y$. Since the procedure considers all three trees in the multi-tree embedding this guarantees the first invariant, i.e., that $w_y = \textsc{MultiTreeDist}(y, S \cup \{x\})^2$ for every $y \in P$ at the end of the procedure. The second invariant is guaranteed by Step 8 which updates all the nodes in the sample-tree so as to satisfy that invariant. $\textsc{MultiTreeOpen}$ therefore updates the data structure to satisfy the invariants. As the distance from a point $x$ to the centers is updated $O(\log(d\Delta))$ times and each time the sample-tree is

updated in time $O(\log n)$ (its height), we have the following running time (see Appendix B for a formal argument).

**Lemma 4.1** *The running time of opening any set $S$ of $k$ points (using calls to MULTITREEOPEN) is $O(n \log(d\Delta) \log n)$.*

Having described how to open a new center, we proceed to describe the simpler algorithm for generating a sample. The pseudo-code of MULTITREESAMPLE is given in Algorithm 2. MULTI-TREESAMPLE traverses the sample-tree from the root to a random leaf by, at each intermediate node, randomly choosing one of its two children proportional to its weight. As the weight of each node in the sample-tree, equals the sum of weights of the points in its subtree (by the second invariant), this guarantees that a point $x$ is sampled with probability $w_x / \sum_{y \in P} w_y$, i.e., proportional to its weight. By the first invariant, this corresponds to sampling from the $D^2$-distribution with respect to the multi-tree distances[8]. Furthermore, the running time of MULTITREESAMPLE is $O(\log n)$ since the height of the sample-tree is $O(\log n)$. (Recall that the sample-tree is a balanced binary tree with $n$ leafs and is thus of height $O(\log n)$. Recall also that the sample-tree is a different tree from the tree embeddings) We summarize these properties of MULTITREESAMPLE in the following lemma.

---

**Algorithm 2** MULTITREESAMPLE

1: Let $v$ be the root of the sample-tree.
2: **while** $v$ is not a leaf **do**
3:     Let $w(L)$ and $w(R)$ be the weight of its left and right child, respectively.
4:     Update $v$ to be its left child with probability $\frac{w(L)}{w(L)+w(R)}$ and to be its right child with remaining probability $\frac{w(R)}{w(L)+w(R)}$.
5: **end while**
**Output:** the point $x$ corresponding to the leaf $v$.

---

**Algorithm 3** FASTK-MEANS++

**Input:** Set of points $P$, number of centers $k$.
1: Set $S \leftarrow \emptyset$
2: MULTITREEINIT ()
3: **while** $|S| < k$ **do**
4:     $x \leftarrow$ MULTITREESAMPLE()
5:     $S \leftarrow S \cup x$
6:     MULTITREEOPEN($x$)
7: **end while**
**Output:** $S$

---

**Lemma 4.2** *Let $S$ be the set of opened centers (using calls to MULTITREEOPEN). Then MULTITREESAMPLE runs in time $O(\log n)$ and each point $x \in P$ is output with probability $\frac{\text{MULTITREEDIST}(x,S)^2}{\sum_{y \in P} \text{MULTITREEDIST}(y,S)^2}$.*

**FASTK-MEANS++.** We can now present a fast algorithm for the $k$-means problem (see Algorithm 3) that samples each center from the $D^2$-distribution with respect to the distances given by the multi-tree embedding. In the next section we show how to adapt the procedure so as to sample from the original $D^2$-distribution by using rejection sampling. The running time directly follows from that, the time to initialize the multi-tree embedding is $O(nd \log(d\Delta))$, the time to initialize the data structure used by MULTITREEOPEN and MULTITREESAMPLE is $O(n \log(d\Delta))$, the total running time of MULTITREEOPEN is $O(n \log(d\Delta) \log n)$ (Lemma 4.1) and the running time of each call to MULTITREESAMPLE is $O(\log n)$ (Lemma 4.2).

**Corollary 4.3** *The running time of FASTK-MEANS++ is $O(nd \log(d\Delta) + n \log(d\Delta) \log n)$.*

## 5 Rejection Sampling Algorithm

In this section we present an algorithm, REJECTIONSAMPLING, that efficiently samples arbitrarily close to the $D^2$-distribution in the original metric. The algorithm is rather simple and its pseudo-code is given in Algorithm 4. The main idea is to use the multi-tree embedding to sample candidate centers but then adjust the sampling probability using rejection sampling.

As for K-MEANS++, the first center that we pick is chosen uniformly at random among all the points. For the rest of the $k - 1$ centers, the idea is to sample a point $x$ using MULTITREESAMPLE, i.e.,

---

[8]We remark that the idea of sampling in this way from a tree has been used in the context of constructing a coreset in [1] (however, their tree depends on a partition of the data and is not necessarily balanced).

form the $D^2$-distribution with respect to the multi-tree distances. Then we open $x$ as a new center with probability proportional to its actual distance to the set of centers in the original metric over the distance in the multi-tree embedding. We repeat this procedure until we pick the rest of the $k-1$ centers. Interestingly, this rejection-sampling procedure guarantees that we sample each of the centers according to the actual $D^2$-distribution. However, the running time of this procedure is of $\Omega(k^2)$ since, for each point $x$ that we sample from the multi-tree, we have to find the closest open center which takes time $\Omega(k)$. In order to improve this running time, we use an approximate nearest neighbor data structure to approximate the distance between $x$ and the closest open center. This enables us to improve the running time to be near linear. The data structure that we use is based on the locality-sensitive hash (LSH) functions developed for Euclidean metrics [3]. We only need to slightly modify their data structure to guarantee monotonicity as we explain in Appendix D.

**Theorem 5.1 (LSH data structure)** *For any set $P$ of $n$ points in $R^d$ and any parameter $c > 1$, there exists a data structure with operations Insert and Query that, with probability at least $1 - 1/n$, have the following guarantees: (i) Insert(p):* Inserts *point $p \in P$ to the data structure in time* $O\left(d\log(\Delta) \cdot (n\log(\Delta))^{O(1/c^2)}\right)$. *(ii) Query(p): Returns a point $q$ that has been inserted into the data structure that is at distance at most $c \cdot \delta$ from $p$, where $\delta$ is the minimum distance from $p$ to a point inserted to the data structure. The query time is* $O\left(d\log(\Delta) \cdot (n\log(\Delta))^{O(1/c^2)}\right)$.
*Furthermore, the data structure is* monotone under insertions*: the distance between $p$ and Query(p) is non-increasing after inserting more points.*

We say that the data structure is successful if the above guarantees hold. By the theorem statement, we know that the data structure is successful with probability at least $1 - 1/n$. The small failure probability will not impact the expected cost of our solution[9]. We therefore *assume throughout the analysis that our data structure is successful*. In Algorithm 4 we present the pseudocode for our algorithm.

In the REJECTIONSAMPLING algorithm (Algorithm 4), the probability on Line 5 is not defined for the case that $S$ is an empty set, i.e., the first iteration of the loop. In this case we assume that this probability is one and the sampled element will be added to $S$. We start be presenting a few properties of REJECTIONSAMPLING algorithm. We show that the expected number of the times that the loop (Line 3) repeats is $O(c^2 d^2 k)$. To that end, we first show that the probability of opening a center in $x$ in any iteration is independent of the MULTITREE embedding and only depends on the LSH data structure. This holds, intuitively, because when we sample a point $x$ by calling MULTITREESAMPLE() we then decide to add it based on the distance to

---

**Algorithm 4** REJECTIONSAMPLING

**Input:** Set of points $P$, number of centers $k$
1: Set $S \leftarrow \emptyset$
2: MULTITREEINIT ()
3: **while** $|S| < k$ **do**
4:     $x \leftarrow$ MULTITREESAMPLE()
5:     **With** **probability** $\min\{1, \frac{\text{DIST}(x,\text{Query}(x))^2}{c^2 \cdot \text{MULTITREEDIST}(x,S)^2}\}$ **do**
6:         $S \leftarrow S \cup x$
7:         MULTITREEOPEN $(x)$
8:         Insert(x)
9: **end while**
**Output:** $S$

---

the point reported by the LSH data structure which removes the dependency on MULTITREEINIT.

Specifically, each point $x$ is first sampled w.p. $\frac{\text{MULTITREEDIST}(x,S)^2}{\sum_{y \in P} \text{MULTITREEDIST}(y,S)^2}$ and then added to set $S$ w.p. $\frac{\text{DIST}(x,\text{Query}(x))^2}{c^2 \cdot \text{MULTITREEDIST}(x,S)^2}$. Therefore, the probability of adding $x$ to $S$ is proportional to $\text{DIST}(x,\text{Query}(x))^2$ and we get (see Appendix C for a formal proof):

**Lemma 5.2** *The probability of inserting a point $x$ to set $S$ in* REJECTIONSAMPLING *algorithm is independent of* MULTITREEINIT *and is equal to $1/n$ for the first iteration and* $\frac{\text{DIST}(x,Query(x))^2}{\sum_{y \in P} \text{DIST}(y,Query(y))^2}$ *for other iterations.*

---

[9]To be completely formal: if we repeat our algorithm for $\log_n(4n\Delta^2)$ times, then we know that with probability at least $1 - 1/(4n\Delta^2)$ one of the runs is with a successful data structure. As squared-distances are at most MAXDIST$^2$ and at least MAXDIST$^2/(2\Delta)^2$, the total cost of a solution with a single opened center is at most $n \cdot$ MAXDIST$^2$. Therefore, the small failure probability of $1/(4n\Delta^2)$ will not have a measurable impact on the expected cost of the best found clustering.

| Algorithm | $k = 100$ | $k = 500$ | $k = 1000$ | $k = 2000$ | $k = 3000$ | $k = 5000$ |
|---|---|---|---|---|---|---|
| FASTK-MEANS++ | 1.0x | 1.0x | 1.0x | 1.0x | 1.0x | 1.0x |
| REJECTIONSAMPLING | 1.04x | 1.09x | 1.04x | 1.07x | 1.01x | 1.28x |
| K-MEANS++ | 0.66x | 3.11x | 6.58x | 15.26x | 18.58x | 42.64x |
| AFKMC2 | 0.89x | 1.88x | 3.80x | 8.5x | 16.61x | 38.7x |

Table 1: Running time of the algorithms divided by the running time of FASTK-MEANS++ for the KDD-Cup dataset. This shows the speed-up that we achieve compared to the K-MEANS++ and AFKMC2.

| Algorithm | $k = 100$ | $k = 500$ | $k = 1000$ | $k = 2000$ | $k = 3000$ | $k = 5000$ |
|---|---|---|---|---|---|---|
| FASTK-MEANS++ | 1.0x | 1.0x | 1.0x | 1.0x | 1.0x | 1.0x |
| REJECTIONSAMPLING | 0.99x | 1.03x | 0.98x | 1.03x | 1.04x | 1.04x |
| K-MEANS++ | 0.76x | 4.55x | 8.89x | 16.98x | 23.03x | 46.26x |
| AFKMC2 | 0.62x | 1.02x | 1.35x | 2.81x | 4.98x | 8.71x |

Table 2: Running time of the algorithms divided by the running time of FASTK-MEANS++ for the Song dataset. This shows the speed-up that we achieve compared to the K-MEANS++ and AFKMC2.

The main ingredient in the running time analysis is to bound the number of repetitions of the loop (Line 3). This is roughly done by arguing that the probability that we add an element $x$ to $S$ after its sampled using MULTITREESAMPLE is $\Omega(\frac{1}{c^2 d^2})$ in expectation. Indeed, from Lemma 3.1 we expect that $\text{MULTITREEDIST}(x, S)^2 \leq O(d^2 \text{DIST}(x, \text{Query}(x))^2)$, so $\frac{1}{\Omega(c^2 d^2)} \leq \frac{\text{DIST}(x, \text{Query}(x))^2}{c^2 \cdot \text{MULTITREEDIST}(x, S)^2}$. Therefore the probability of passing Line 5 is at least $\frac{1}{\Omega(c^2 d^2)}$. It follows that, in expectation, $O(c^2 d^2 k)$ repetitions suffices to add $k$ points to $S$. The formal proof is presented in Appendix E.1.

**Lemma 5.3** *The expected number of the times that the loop (Line 3) is repeated is of $O(c^2 d^2 k)$.*

Putting the discussed ingredients and the approximation ratio analysis together, we get the following result, the proof is presented in Appendix E.1.

**Theorem 5.4** *For any constant $c > 1$, with probability at least $(1 - 1/n)$ REJECTIONSAMPLING always samples points $x$ that are at most a factor $c^2$ away from the $D^2$-distribution, its expected running time is $O\left(n \log(d\Delta)(d + \log n) + kc^2 d^3 \log(\Delta) \cdot (n \log(\Delta))^{O(1/c^2)}\right)$, and it returns a solution that in expectation is a $O(c^6 \log k)$-approximation of the optimal solution.*

We remark that the runtime can be improved in the case of a large $d$ by first applying a dimensionality reduction [8, 26] that reduces the dimension of the input points to $O(\log n)$ in time $O(nd \log n)$ and maintains the cost of any clustering up to a constant factor. These works actually prove that the dimension can be reduced to $O(\log k)$. However, by using $O(\log n)$ our algorithm can output the solution for *all* $k = 1, 2, \ldots, n$ in near-linear running time $\widetilde{O}\left(nd + n \log \Delta + c^2 k \log(\Delta)(n \log \Delta)^{O(1/c^2)}\right)$ (where $\widetilde{O}$ suppresses logarithmic terms in $n$) while maintaining the same asymptotic approximation guarantee as K-MEANS++. Selecting $\varepsilon = O(1/c^2)$ then yields the following

**Corollary 5.5** *For $\varepsilon > 0$, there is an $O_\varepsilon(\log k)$-approximation algorithm for the k-means problem with a running time of $\widetilde{\Theta}(nd + (n \log(\Delta))^{1+\varepsilon})$.*

# 6   Empirical Evaluation

In this section we empirically validate out theoretical results by comparing our algorithms FASTK-MEANS++ and REJECTIONSAMPLING (see the details on how we set the parameters for LSH in Appendix D.3 ) with the following two baselines:

K-MEANS++ algorithm: Perhaps the most commonly used algorithm in this field. It samples $k$ points according to the $D^2$-distribution.

AFKMC2 algorithm: A recent result [5] based on random walks that improves the running time of the K-MEANS++ algorithm while maintaining a (weaker) theoretical guarantee on the solution quality.

| Algorithm | $k = 100$ | $k = 500$ | $k = 1000$ | $k = 2000$ | $k = 3000$ | $k = 5000$ |
|---|---|---|---|---|---|---|
| FASTK-MEANS++ | 1.0x | 1.0x | 1.0x | 1.0x | 1.0x | 1.0x |
| REJECTIONSAMPLING | 0.98x | 1.26x | 1.17x | 1.07x | 1.0x | 0.95 |
| K-MEANS++ | 0.89x | 4.78x | 8.92x | 14.18x | 23.57x | 36.69 |
| AFKMC2 | 0.76x | 0.77x | 1.12x | 1.15x | 1.54x | 2.56x |

Table 3: Running time of the algorithms divided by the running time of FASTK-MEANS++ for the Census dataset. This shows the speed-up that we achieve compared to the K-MEANS++ and AFKMC2.

| Algorithm | $k = 100$ | $k = 500$ | $k = 1000$ | $k = 2000$ | $k = 3000$ | $k = 5000$ |
|---|---|---|---|---|---|---|
| FASTK-MEANS++ | 30335 | 5771 | 2957 | 1582 | 1070 | 640 |
| REJECTIONSAMPLING | 29243 | 5857 | 2999 | 1581 | 1095 | 642 |
| K-MEANS++ | 24552 | 5128 | 2695 | 1423 | 968 | 562 |
| AFKMC2 | 25598 | 5384 | 2883 | 1512 | 1045 | 622 |
| UNIFORMSAMPLING | 148594 | 51692 | 26199 | 15927 | 13922 | 10017 |

Table 4: Costs of the solutions produced by the algorithm for KDD-Cup dataset for various values of $k$. All the numbers are scaled down by a factor $10^3$.

**Datasets, Experiments, and Setup**  We ran our algorithms on three classic datasets from UCI library [19]: KDD-Cup [14] ($311,029$ points of dimension 74) and song [9] ($515,345$ points of dimension 90) Census [24] ($2,458,285$ points of dimension 68). We did not apply any dimensionality reduction technique for any of the algorithms; all the considered data sets is of small dimension. We compare the quality of the clustering, i.e., the cost of the objective function, along with their running times. For the AFKMC2 algorithm, we used the code provided by the authors with the same parameter suggested there, i.e., $m = 200$.[10] The algorithms were run on a standard desktop computer.

**Discussion**  Our results show that the algorithms we propose are much faster than both the baselines, i.e., K-MEANS++ and AFKMC2, as $k$ grows. For large $k = 5000$, it is an order of magnitude faster than both K-MEANS++ and AFKMC2. Moreover, the running time of our algorithms is already significantly faster than both baselines for moderate values of $k$ such as $k = 500$ for KDD-Cup and $k = 1000$ for Song and Census. We refer to Tables 1, 2, and 3 for more details.

Importantly, we achieve this improvement in the running time without making any significant sacrifice to solution quality from both a theoretical and experimental perspective. While the solution quality is sometimes worse by 10-15% for small $k$, the $k$-means costs of the solutions produced by FASTK-MEANS++ and REJECTIONSAMPLING algorithms are comparable (overall almost the same) with the baselines for all the experiments for moderate values of $k \geq 1000$. This is in contrast to the simplest seeding algorithm UNIFORMSAMPLING which selects the $k$ centers uniformly at random from the input data set. While UNIFORMSAMPLING clearly provides for a very fast seeding algorithm, it does so by significantly deteriorating the solution quality. This can e.g. be seen in our results for the KDD-Cup dataset where UNIFORMSAMPLING consistently gives solutions of much worse quality. For more details, see Tables 4, 5, and 6 where the solution costs are given. The variance along with experimental setting is reported in Appendix F.

## 7 Conclusions

In this paper we present new efficient algorithms for $k$-means++ seeding. Our algorithms outperform previous work as $k$ grows and come with strong theoretical guarantees. Interesting avenues for future work are to develop efficient distributed algorithms for the same problem and to prove lower bounds on the running time.

---

[10]$m$ is the number of steps in the random walk.

| Algorithm | $k = 100$ | $k = 500$ | $k = 1000$ | $k = 2000$ | $k = 3000$ | $k = 5000$ |
|---|---|---|---|---|---|---|
| FASTK-MEANS++ | 21898668 | 16732379 | 14987614 | 13477854 | 12691185 | 11628744 |
| REJECTIONSAMPLING | 21743137 | 16851767 | 15024812 | 13558210 | 12720314 | 11654493 |
| K-MEANS++ | 21583261 | 16409834 | 14746899 | 13395052 | 12480900 | 11496421 |
| AFKMC2 | 21596184 | 16344430 | 14750601 | 13246450 | 12450688 | 11476712 |
| UNIFORMSAMPLING | 23255642 | 17919981 | 16373134 | 14579718 | 13934375 | 12938255 |

Table 5: Costs of the solutions produced by the algorithms for the Song dataset for various values of $k$. All the numbers are scaled down by a factor $10^5$.

| Algorithm | $k = 100$ | $k = 500$ | $k = 1000$ | $k = 2000$ | $k = 3000$ | $k = 5000$ |
|---|---|---|---|---|---|---|
| FASTK-MEANS++ | 17304 | 9820 | 7883 | 6326 | 5625 | 4868 |
| REJECTIONSAMPLING | 17735 | 9970 | 8031 | 6432 | 5644 | 4893 |
| K-MEANS++ | 18498 | 9585 | 7812 | 6254 | 5561 | 4815 |
| AFKMC2 | 17242 | 9844 | 7710 | 6272 | 5595 | 4838 |
| UNIFORMSAMPLING | 19912 | 10630 | 8678 | 6880 | 6120 | 5228 |

Table 6: Costs of the solutions produced by the algorithm for Census dataset for various values of $k$. All the numbers are scaled down by a factor $10^4$.

## Broader Impact

Our work focuses on speeding-up the very popular K-MEANS++ algorithm for clustering. The K-MEANS++ algorithm is used in a variety of domains and is an important tool for extracting information, compressing data, or unsupervised classification tasks. Our result shows that one can obtain a much faster implementation of the $k$-means++ algorithm while preserving its approximation guarantees both in theory and in practice. Therefore, we expect that our new algorithm could have impact in several domains in which clustering plays an important role. A broader concrete impact in society is harder to predict since this is mainly fundamental research.

## Acknowledgments and Disclosure of Funding

## References

[1] Marcel R Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler. Streamkm++ a clustering algorithm for data streams. *Journal of Experimental Algorithmics (JEA)*, 17:2–1, 2012.

[2] Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 61–72. IEEE Computer Society, 2017.

[3] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *2006 47th annual IEEE symposium on foundations of computer science (FOCS'06)*, pages 459–468. IEEE, 2006.

[4] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[5] Olivier Bachem, Mario Lucic, Hamed Hassani, and Andreas Krause. Fast and provably good seedings for k-means. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 55–63. Curran Associates, Inc., 2016.

[6] Olivier Bachem, Mario Lucic, S. Hamed Hassani, and Andreas Krause. Approximate $k$-means++ in sublinear time. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 1459–1467, 2016.

[7] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 184–193, 1996.

[8] Luca Becchetti, Marc Bury, Vincent Cohen-Addad, Fabrizio Grandoni, and Chris Schwiegelshohn. Oblivious dimension reduction for k-means – beyond subspaces and the johnson-lindenstrauss lemma. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, 2019.

[9] Thierry Bertin-Mahieux, Daniel PW Ellis, Brian Whitman, and Paul Lamere. The million song dataset. 2011.

[10] Aditya Bhaskara and Maheshakya Wijewardena. Distributed clustering via lsh based data partitioning. In *International Conference on Machine Learning*, pages 570–579, 2018.

[11] Anup Bhattacharya, Jan Eube, Heiko Röglin, and Melanie Schmidt. Noisy, greedy and not so greedy k-means++. *CoRR*, abs/1912.00653, 2019.

[12] Thomas Bottesch, Thomas Bühler, and Markus Kächele. Speeding up k-means by approximating euclidean distances via block vectors. In *International Conference on Machine Learning*, pages 2578–2586, 2016.

[13] Ke Chen. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM Journal on Computing*, 39(3):923–947, 2009.

[14] KDD Cup. Protein homology dataset, 2004.

[15] Ryan R Curtin. A dual-tree algorithm for fast k-means clustering with large k. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 300–308. SIAM, 2017.

[16] Sanjoy Dasgupta. Lecture 3 – Algorithms for k-means clustering, 2013. Accessed January 7, 2020.

[17] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

[18] Yufei Ding, Yue Zhao, Xipeng Shen, Madanlal Musuvathi, and Todd Mytkowicz. Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup. In *International Conference on Machine Learning*, pages 579–587, 2015.

[19] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[20] Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 291–300, 2004.

[21] Oktie Hassanzadeh, Fei Chiang, Renée J. Miller, and Hyun Chul Lee. Framework for evaluating clustering algorithms in duplicate detection. *PVLDB*, 2(1):1282–1293, 2009.

[22] Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 373–380, 2004.

[23] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing (STOC)*, pages 604–613, 1998.

[24] Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In Evangelos Simoudis, Jiawei Han, and Usama M. Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, pages 202–207. AAAI Press, 1996.

[25] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129 – 137, 1982.

[26] K. Makarychev, Y. Makarychev, and Ilya Razenshteyn. Performance of johnson—lindenstrauss transform for k-means and k-medians clustering. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, 2019.

[27] James Newling and François Fleuret. Fast k-means with accurate bounds. In *International Conference on Machine Learning*, pages 936–944, 2016.

[28] Feng Qian, Abhinav Pathak, Yu Charlie Hu, Zhuoqing Morley Mao, and Yinglian Xie. A case for unsupervised-learning-based spam filtering. In *Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 367–368, 2010.

[29] M. Ali Rostami, Alieh Saeedi, Eric Peukert, and Erhard Rahm. Interactive visualization of large similarity graphs and entity resolution clusters. In *Proceedings of the 21th International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018.*, pages 690–693, 2018.

[30] Mina Sheikhalishahi, Andrea Saracino, Mohamed Mejri, Nadia Tawbi, and Fabio Martinelli. Fast and effective clustering of spam emails based on structural similarity. In *International Symposium on Foundations and Practice of Security*, pages 195–211. Springer, 2015.

# A   Proofs Omitted from Section 3

**Lemma 3.1** *For any point $p$, and set $S$, we have* $\text{DIST}(p, S)^2 \leq \text{MULTITREEDIST}(p, S)^2$ *and* $\mathbf{E}[\text{MULTITREEDIST}(p, S)^2] \leq O(d^2 \cdot \text{DIST}(p, S)^2)$.

*Proof.* We start by showing that $\text{DIST}(p, S)^2 \leq \text{MULTITREEDIST}(p, S)^2$. We prove this for every single tree, which implies the result. In particular, we show that for any tree and for any two points $p, q \in P$, we have $\text{DIST}(p, q) \leq \text{TREEDIST}(p, q)$. Assume that the lowest common ancestor of $p, q$ is at height $i$. Therefore in each dimension, they differ at most by $\frac{2 \cdot \text{MAXDIST}}{2^i}$ since the side length of the cube at this height is $\frac{2 \cdot \text{MAXDIST}}{2^i}$, therefore

$$\text{DIST}(p, q) \leq \sqrt{d} \cdot \frac{2 \cdot \text{MAXDIST}}{2^i}.$$

Moreover, $\text{TREEDIST}(p, q)$ is defined as the length of the shortest path between them and the length of the edge at height $j$ is $\sqrt{d} \cdot \frac{\text{MAXDIST}}{2^j}$ for $0 \leq j < H$. So

$$\text{TREEDIST}(p, q) = 2 \sum_{i \leq j < H} \sqrt{d} \cdot \frac{\text{MAXDIST}}{2^j} = 2\sqrt{d} \cdot \text{MAXDIST} \sum_{i \leq j < H} 2^{-j} \geq \sqrt{d} \cdot \frac{2 \cdot \text{MAXDIST}}{2^i}.$$

Therefore, $\text{DIST}(p, q) \leq \sqrt{d} \cdot \frac{2 \cdot \text{MAXDIST}}{2^i} \leq \text{TREEDIST}(p, q)$, which concludes the proof of the first part of the lemma.

Now we focus on the second part of the lemma, i.e., $\mathbf{E}[\text{MULTITREEDIST}(p, q)^2] \leq O(d^2 \cdot \text{DIST}(p, q)^2)$. Let $p = (p_1, \ldots, p_d)$ and $q = (q_1, \ldots, q_d)$ be two points in $\mathbb{R}^d$. We first analyze the probability that these two points are separated at a certain height $i$ in a single tree. In a single tree, two points are separated at height $i$ if they are separated in at least one of the coordinates. The probability that $p$ and $q$ are separated in the $j$-th dimension is at most $\frac{|p_j - q_j|}{\frac{2 \cdot \text{MAXDIST}}{2^i}}$. Therefore, if we let $s_i$ denote the probability that they are separated at height $i$ (but at no smaller height), then by the union bound $s_i \leq \sum_{1 \leq j \leq d} \frac{|p_j - q_j|}{\frac{2 \cdot \text{MAXDIST}}{2^i}} \leq \sqrt{d} \cdot \frac{2^i \text{DIST}(p,q)}{2 \cdot \text{MAXDIST}}$, where the second inequality holds because $\sum_{1 \leq j \leq d} |p_j - q_j| \leq \sqrt{d} \cdot \text{DIST}(p, q)$. So the probability that they are separated at height $i$ or before is at most

$$\sum_{0 \leq j \leq i} s_j \leq \sum_{0 \leq j \leq i} \sqrt{d} \cdot \frac{2^j \text{DIST}(p,q)}{2 \cdot \text{MAXDIST}} \leq \sqrt{d} \cdot \frac{2^{i+1} \text{DIST}(p,q)}{2 \cdot \text{MAXDIST}}.$$

Notice that, as before, we also have that $\text{TREEDIST}(p, q) = 2\sqrt{d} \cdot \text{MAXDIST} \cdot \sum_{i \le j < H} 2^{-j} \le \sqrt{d} \cdot \frac{4 \cdot \text{MAXDIST}}{2^i}$ in the case that $p, q$ are separated at height $i$. Now recall that $\text{MULTITREEDIST}$ is the minimum distance among all the three tree embeddings, so it is enough that the two points are separated at height $i$ in a single tree to be at this distance (in the other two trees they can be separated in a level closer to the root). There are three ways to select the tree of minimum distance and so

$$
\begin{aligned}
\mathbf{E}[\text{MULTITREEDIST}(p, q)^2] &\le 3 \sum_{0 \le i < H} s_i \left( \sum_{0 \le j \le i} s_j \right)^2 \cdot \left( \sqrt{d} \cdot \frac{4 \cdot \text{MAXDIST}}{2^i} \right)^2 \\
&\le 3 \sum_{0 \le i < H} s_i \left( \sqrt{d} \cdot \frac{2^{i+1} \text{DIST}(p, q)}{2 \cdot \text{MAXDIST}} \right)^2 \cdot \left( \sqrt{d} \cdot \frac{4 \cdot \text{MAXDIST}}{2^i} \right)^2 \\
&= 3 \sum_{0 \le i < H} s_i 16 d^2 \text{DIST}(p, q)^2 = 48 d^2 \text{DIST}(p, q)^2 \sum_{0 \le i < H} s_i \\
&\le 48 d^2 \text{DIST}(p, q)^2 = O(d^2 \text{DIST}(p, q)^2),
\end{aligned}
$$

where the last inequality holds because $\sum s_i \le 1$ since the $s_i$'s denote the probabilities of mutually disjoint events. $\qquad\square$

## B   Proof Omitted from Section 4

**Lemma 4.1** *The running time of opening any set $S$ of $k$ points (using calls to $\text{MULTITREEOPEN}$) is $O(n \log(d\Delta) \log n)$.*

*Proof.* As the multi-tree embedding consists of three trees, it is sufficient to analyze the running time of the for-loop at Step 1 for a single tree $T$. In a single call to $\text{MULTITREEOPEN}$ we have that Steps 2- 4 runs in time $O(\log(d\Delta))$ since each tree in the multi-tree embedding has depth at most $H = \log(d\Delta)$. Hence the total running time for these steps across the $k$ calls to $\text{MULTITREEOPEN}$ is $O(k \log(d\Delta))$.

The for-loop at Step 5 can be implemented as follows. Observe that the distance $\text{TREEDIST}_T(x, y)$ for a point $y \in P_T(v_0)$ equals 0. Furthermore, for $i = 1, 2, \ldots, \ell$ and a point $y \in P_T(v_i) \setminus P_T(v_{i-1})$, the distance $\text{DIST}_T(x, y)$ equals twice the length of the path in $T$ from $v_0$ to $v_i$. We can thus calculate all relevant distances in time $O(\log(d\Delta))$ (and in time $O(k \log(d\Delta))$ across all $k$ calls).

The weights of the points can then be updated by first considering the points in $P_T(v_0)$, then those in $P_T(v_1) \setminus P_T(v_0)$, and so on until the points in $P_T(v_\ell) \setminus P_T(v_{\ell-1})$. The running time of Step 7 is thus $O(\sum_{i=0}^{\ell} |P_T(v_i)|)$. Moreover, as each execution of Step 8 takes time $O(\log n)$ (since the sample-tree is balanced binary tree with $n$ leaves and thus of height $O(\log n)$), the running time of this step is $O(\sum_{i=0}^{\ell} |P_T(v_i)| \cdot \log n)$. Now a key point is that a node in a tree in the multi-tree embedding can only be marked once. Therefore, using that $\sum_v |P_T(v)| = O(n \log(d\Delta))$, we have that the total running time of the for-loop is $O(n \log(d\Delta) \log n)$. The total running time of the $k$ calls to $\text{MULTITREEOPEN}$ is therefore $O(k \log n + n \log(d\Delta) \log n)$. $\qquad\square$

## C   Proofs Omitted from Section 5

**Lemma 5.2** *The probability of inserting a point $x$ to set $S$ in $\text{REJECTIONSAMPLING}$ algorithm is independent of $\text{MULTITREEINIT}$ and is equal to $1/n$ for the first iteration and $\frac{\text{DIST}(x, Query(x))^2}{\sum_{y \in P} \text{DIST}(y, Query(y))^2}$ for other iterations.*

*Proof.* The proof is by induction on the size of the set $S$. The base case is clear since, as aforementioned, in Line 5 we accept the point with probability one and $\text{MULTITREESAMPLE}$ () returns each point with probability $1/n$. Now assume that lemma holds until $\ell$ centers are added (i.e., $|S| = \ell$). It follows from the induction hypothesis the current set $S$ is independent from the $\text{MULTITREE}$ initialization since all the elements added so far are independent. By the assumption that

the data structure is successful, the above minimum on Line 5 is always attained by the second term, i.e., we have $\frac{\text{DIST}(x,\text{Query}(x))^2}{c^2 \cdot \text{MULTITREEDIST}(x,S)^2} \leq 1$. Indeed, $\text{DIST}(x, \text{Query(x)})^2 \leq c^2 \cdot \text{DIST}(x, S_i)^2 \leq c^2 \cdot \text{MULTITREEDIST}(x, S_i)^2$, where the first inequality is by the success of the data structure and the second inequality is by the fact that the multi-tree embedding only increases distances (see Lemma 3.1).

Therefore in REJECTIONSAMPLING algorithm, the probability that a point $x$ is sampled is

$$\frac{\text{MULTITREEDIST}(x, S)^2}{\sum_{y \in P} \text{MULTITREEDIST}(y, S)^2} \cdot \frac{\text{DIST}(x, \text{Query}(x)^2)}{c^2 \cdot \text{MULTITREEDIST}(x, S)^2}$$

$$= \frac{\text{DIST}(x, \text{Query}(x))^2}{c^2 \cdot \sum_{y \in P} \text{MULTITREEDIST}(y, S)^2} \cdot$$

Notice that the denominator does not depend on the point $x$ and one can think of it as a constant term. We repeat the sampling process until we pick a point. Therefore, the probability of choosing any point $x$ is $\frac{\text{DIST}(x,\text{Query}(x))^2}{\sum_{y \in P} \text{DIST}(y,\text{Query}(y))^2}$, independent of the the MULTITREE initialization. This completes the inductive step and concludes the proof of the lemma. $\qquad\square$

**Lemma 5.3** *The expected number of the times that the loop (Line 3) is repeated is of $O(c^2 d^2 k)$.*

*Proof.* Let $R$ be the random variable that equals the number of times that the loop is repeated. We further divide $R$ into $R_0, \ldots, R_{k-1}$ where, for $i \in \{0, \ldots, k-1\}$, $R_i$ denotes the number of times the loop is executed when the set $S$ of opened centers has size $i$, i.e., when exactly $i$ centers have been opened. Then $R = R_0 + R_1 + \ldots + R_{k-1}$ and, by linearity of expectation,

$$\mathbf{E}[R] = \mathbf{E}[R_0] + \mathbf{E}[R_1] + \ldots + \mathbf{E}[R_{k-1}].$$

We have $\mathbf{E}[R_0] = 1$ since the first center is selected uniformly at random and it is always opened, i.e., added to $S$. We complete the proof by proving

$$\mathbf{E}[R_i] \leq O(c^2 \cdot d^2) \qquad \text{for } i \in \{1, \ldots, k-1\}.$$

Consider $R_i$ and let $S_i$ be the set containing the first $i$ centers that were opened by the algorithm. We actually prove the stronger statement that $\mathbf{E}[R_i] \leq O(c^2 \cdot d^2)$ no matter the set $S_i$.

Consider an iteration of the loop. First, as argued in the proof of Lemma 5.2, the probability that an iteration of the loop results in adding a point $x$ to the set of opened centers equals $\frac{1}{c^2} \cdot \frac{\sum_{x \in P} \text{DIST}(x,\text{Query}(x))^2}{\sum_{y \in P} \text{MULTITREEDIST}(y,S_i)^2}$. If we let $q$ denote this probability then $\mathbf{E}[R_i] = \sum_{t=1}^{\infty} tq \cdot (1-q)^{t-1}$, which equals $1/q$.

We thus have

$$\mathbf{E}[R_i] = 1/q = c^2 \cdot \frac{\sum_{y \in P} \text{MULTITREEDIST}(y, S_i)^2}{\sum_{x \in P} \text{DIST}(x, \text{Query}(x))^2},$$

for a fixed multi-tree embedding.

The lemma now follows from that $\text{DIST}(x, \text{Query}(x))^2 \geq \text{DIST}(x, S_i)^2$ and from Lemma 5.2 which says that the distribution of the random multi-tree embedding is independent from $S_i$. We thus have, by also taking the expectation over the random multi-tree embedding (see Lemma 3.1), that

$$\mathbf{E}[R_i] \leq c^2 \cdot \frac{O(d^2) \sum_{y \in P} \text{DIST}(y, S_i)^2}{\sum_{x \in P} \text{DIST}(x, \text{Query}(x))^2} = O(c^2 \cdot d^2).$$

$\qquad\square$

# D  LSH data structure

In this section we describe the data structure guaranteed by Theorem 5.1. It follows the construction first introduced in [23]. Their construction is based on locality-sensitive hash families:

**Definition D.1 (Locality-sensitive hashing)** *Let $\mathcal{H}$ be a family of hash functions mapping $\mathbb{R}^d$ to some universe $U$. We say that $\mathcal{H}$ is $(R, cR, p_1, p_2)$-sensitive if* for any $p, q \in \mathbb{R}^d$ *it satisfies the following properties:*

- *If $\|p - q\|_2 \leq R$ then $\mathrm{Pr}_{\mathcal{H}}[h(p) = h(q)] \geq p_1$.*

- *If $\|p - q\|_2 \geq cR$ then $\mathrm{Pr}_{\mathcal{H}}[h(p) = h(q)] \leq p_2$.*

The specific family of hash functions that we use is by [3]. We summarize the main properties of their family in the following theorem. Here, and in the following, we denote by $n$ the size of the data set $P \subseteq \mathbb{R}^d$.

**Theorem D.2 ([3])** *For any $R > 0$ and $c > 1$, there exists a family $\mathcal{H}$ of hash functions for $\mathbb{R}^d$ with the following properties:*

- *$\mathcal{H}$ is $(R, cR, p_1, p_2)$-sensitive with $\frac{\log(1/p_1)}{\log(1/p_2)} = 1/c^2 + o(1)$ and $\frac{1}{\log(1/p_2)} = O(1)$.*

- *The time to compute $h(p)$ for $h \in \mathcal{H}$ and $p \in \mathbb{R}^d$ is $O(dn^{o(1)})$.*

We first describe a data structure for the "gap version". Then we show, using standard arguments, that this gives the data structure as stated in Theorem 5.1.

## D.1  Monotone data structure for gap version

In this section we are going to develop a data structure that is parameterized by $c \geq 1$ (the accuracy) and $R > 0$ (the scale). We refer to it as the $(c, R)$-gap data structure. It is different from the data structure guaranteed by Theorem 5.1 as it only have guarantees that depend on the scaling parameter $R$ (see the statement of Theorem D.3 below).

**Selection of parameters.** We let $\mathcal{H}$ be the $(R, cR, p_1, p_2)$-sensitive hash family for $\mathbb{R}^d$ given by Theorem D.2. We also let $\delta > 0$ be a parameter of our data structure that determines the probability of failure (and impacts the running time). Other parameters that we use are now determined as follows:

- $\eta = \left(\frac{\delta}{n}\right)^{\frac{3}{1-\rho}}$ where $\rho = \frac{\log(1/p_1)}{\log(1/p_2)}$,

- $m = \frac{\log(1/\eta)}{\log(1/p_2)} = O(\log(1/\eta))$, and

- $\ell = 100 \cdot \log(1/\eta) \cdot (1/\eta)^\rho$.

**Description of data structure.** The data structure is based on $\ell$ hash tables $T_1, T_2, \ldots, T_\ell$ (with linked lists at each entry to deal with collisions). The $\ell$ hash functions $f_1, f_2, \ldots, f_\ell$ for these tables are constructed from $\mathcal{H}$ as follows: for $i \in \{1, 2, \ldots, \ell\}$, $f_i$ is obtained by selecting $m$ *independent samples* $h_{i,1}, h_{i,2}, \ldots, h_{i,m}$ from $\mathcal{H}$. That is, $f_i$ is a $m$-dimensional hash function defined by

$$f_i(p) = [h_{i,1}(p), h_{i,2}(p), \ldots, h_{i,m}(p)] \qquad \text{for } p \in \mathbb{R}^d.$$

We are now ready to define the operations Insert and Query:

- Insert($p$): A point $p \in P \subseteq \mathbb{R}^d$ is inserted in each of the $\ell$ hash tables by appending the point at *the end* of the linked list associated to the entry $T_i[f_i(p)]$ for $i = 1, 2, \ldots, \ell$.
- Query($p$): For each $i \in \{1, 2, \ldots, \ell\}$, let $q_i$ be the first element (if any) in the linked list $T_i[f_i(p)]$ that satisfies $\mathrm{DIST}(p, q_i) \leq cR$. This gives up to $\ell$ candidate points, one for each hash table. Among these candidate points, *output the one with the minimum distance to $p$* (or output none if no candidate point is found in any of the hash tables).

This completes the description of the $c$-NN data structure and we proceed to its analysis.

**Analysis.** We show that the described data structure satisfies the following guarantees:

**Theorem D.3** *For a data set $P \subseteq \mathbb{R}^d$ of $n$ points, the data structure with error parameter $\delta > 0$ satisfies the following guarantees:*

1. *The Insert operation runs in time $O\left(d \cdot (n/\delta)^{O(1/c^2)}\right)$.*

2. *With probability at least $1 - \delta$, the Query operation satisfies the following. Given $p \in P$, if there exists an inserted point within distance $R$ from $p$, then Query(p) returns a point $q$ with $\mathrm{DIST}(p,q) \leq cR$. Moreover, the running time is time is $(n/\delta)^{O(1/c^2)}$.*

*Furthermore, the data structure is* monotone under insertions*: the distance between $p$ and Query(p) is non-increasing after inserting more points.*

Throughout the analysis we assume that $c$ is a large enough constant and that $n$ is sufficiently large. This is motivated by the fact that otherwise a trivial data structure can achieve the bounds claimed by the theorem.

The analyses of the monotonicity property and the running time of the insertion operation are rather immediate:

- The monotonicity property is by definition of the operations Insert and Query. To see that, suppose we run Query(p) for a point $p$. We will argue that inserting any new point $p'$ may not increase the distance $\mathrm{DIST}(p, \mathrm{Query}(p))$. Indeed, when $p'$ is inserted it is appended to the end of the linked-lists $T_i[f_i(p)]$ for $i = 1, 2, \ldots, \ell$. Now when we execute Query(p) the only way that $p'$ will be one of the candidate points $q_1, q_2, \ldots, q_\ell$ is if it, for some $i \in \{1, 2, \ldots, \ell\}$, is the first point in $T_i[f_i(p)]$ within distance $cR$ from $p$. It follows (since insertions are appended at the end of the linked-lists whereas queries inspects the lists from the beginning) that all the the points that were candidates before the insertion of $p'$ are still candidates. Therefore the distance from $p$ to the minimum distance point (of the candidates) can only decrease after inserting a new point $p'$.

- We proceed to analyze the running time of the Insert operation. On the insertion of a point $p \in P$, it is appended to each of the $\ell$ linked lists $T_1[f_1(p)], T_2[f_2(p)], \ldots, T_\ell[f_\ell(p)]$. Appending an element to a linked list takes $O(1)$ time whereas the cost of calculating a single hash $f_i(p)$ is $m$ times the cost of calculating $h(p)$ for a single $h \in \mathcal{H}$, which in turn by Theorem D.2 is $O(dn^{o(1)})$. The running time of an insertion is therefore dominated by the time it takes to calculate the $\ell$ hashes $f_1(p), f_2(p), \ldots, f_\ell(p)$, which by the above arguments takes time

$$\ell \cdot m \cdot O(dn^{o(1)}) = O\left(\log(1/\eta) \cdot (1/\eta)^\rho\right) \cdot O\left(\log(1/\eta)\right) \cdot O(dn^{o(1)})$$
$$= O\left(\log(n/\delta)^2 \cdot (n/\delta)^{3\rho/(1-\rho)}\right) \cdot O(dn^{o(1)})$$
$$= O\left(d \cdot (n/\delta)^{O(1/c^2)}\right),$$

where we used that $c$ is a large enough constant for the last equality.

We proceed to analyze the Query operation which requires a little more work. In order to guarantee that Query returns a nearby point if one exists, we need bound the probability of having a false negative. On the other hand, to bound the running time of the Query operation we need to bound the false positives. The following two lemmas bounds these quantities, starting with the probability of false positives.

**Lemma D.4** *For any $i \in \{1, 2, \ldots, \ell\}$ and two points $p, q \in \mathbb{R}^d$ with $\mathrm{DIST}(p, q) \geq cR$, we have*
$$\Pr[f_i(p) = f_i(q)] \leq \eta.$$

*Proof.* By the independence of $h_{i,1}, h_{i,2}, \ldots, h_{i,m}$, we have
$$\Pr[f_i(p) = f_i(q)] = \Pr_{h \sim \mathcal{H}}[h(p) = h(q)]^m \leq p_2^m,$$

which by the selection of $m$ equals $\eta$. ☐

**Lemma D.5** *For any two points $p, q \in \mathbb{R}^d$ with $\text{DIST}(p,q) \leq R$, we have*
$$\Pr[\exists i \mid f_i(p) = f_i(q)] \geq 1 - \eta \,.$$

*Proof.* Similar to the calculations in the proof of the previous lemma, we have
$$\begin{aligned}
\Pr[\exists i \mid f_i(p) = f_i(q)] &= 1 - \Pr[\forall i, f_i(p) \neq f_i(q)] \\
&= 1 - \Pr[f_i(p) \neq f_i(q)]^\ell \\
&\geq 1 - (1 - p_1^m)^\ell \,.
\end{aligned}$$

By the definition of $\rho$, $p_1 = p_2^\rho$ and so
$$\begin{aligned}
\Pr[\exists i \mid f_i(p) = f_i(q)] &\geq 1 - (1 - p_2^{\rho m})^\ell \\
&= 1 - (1 - \eta^\rho)^\ell \\
&\geq 1 - \eta \,,
\end{aligned}$$

where the last inequality is by the selection of $\ell$. ☐

Equipped with these two lemmas we are now ready to analyze the Query operaton. Specifically, we have the following corollary:

**Corollary D.6** *Consider a set $P$ of $n$ points in $\mathbb{R}^d$. Then with probability at least $1 - \delta$ we have that the hash functions $f_1, f_2, \ldots, f_\ell$ satisfy the following:*

- *For any $p, q \in P$ with $\text{DIST}(p,q) \geq cR$, we have $f_i(p) \neq f_i(q)$ for all $i \in \{1, 2, \ldots, \ell\}$.*

- *For any $p, q \in P$ with $\text{DIST}(p,q) \leq R$, we have that there is an $i \in \{1, 2, \ldots, \ell\}$ such that $f_i(p) = f_i(q)$.*

Before giving the proof of the corollary, note that the first property implies that we have no false positives. Therefore, the running time of Query is the same as for Insertion: it is dominated by the time to calculate the $\ell$ hash functions which is $O\left(d \cdot (n/\delta)^{O(1/c^2)}\right)$. Moreover, the second property guarantees that we always have a hash collision when there is a nearby point of the query-point $p$. This implies that Query($p$) returns a point $q$ with $\text{DIST}(p,q) \leq cR$ if there is a point within distance $R$ from $p$ that has been inserted. To complete the proof of Theorem D.3 it thus remains to prove the corollary:

*Proof of Corollary.* We show that each of the two properties hold fail probability at most $\delta/2$. The statement then follows by the union bound.

For the first property, there are $\ell$ hash functions and at most $\binom{n}{2} \leq n^2$ pairs $p, q \in P$ such that $\text{DIST}(p,q) \geq cR$. Therefore, by Lemma D.4 and the union bound, we have that the first property fails with probability at most
$$\begin{aligned}
\ell \cdot n^2 \cdot \eta &= (100 \cdot \log(1/\eta) \cdot (1/\eta)^\rho) \cdot n^2 \cdot \eta \\
&= (100 \cdot \log(1/\eta)) \cdot n^2 \cdot \eta^{1-\rho} \\
&= \left(\frac{300}{1-\rho} \cdot \log(n/\delta)\right) \cdot n^2 \cdot (\delta/n)^3 \\
&\leq \delta/2 \,,
\end{aligned}$$

where for the last inequality we used that $n$ and $c$ are large.

For the second property, there are at most $\binom{n}{2} \leq n^2$ pairs $p, q \in P$ such that $\text{DIST}(p,q) \leq R$. So by the union bound and Lemma D.5, we have that the second property fails with probability at most $n^2 \cdot \eta$ which by the above calculations is at most $\delta/2$. ☐

## D.2 Putting everything together: Proof of Theorem 5.1

The proof of Theorem 5.1 now follows from Theorem D.3 by standard arguments. Again we assume that $c$ is a large constant (since otherwise a trivial data structure will satisfy the properties of the theorem).

Recall that all distances are between $\text{MAXDIST}/(2\Delta)$ and $\text{MAXDIST}$. We make $\log(2\Delta)$ many copies of the gap data structure guaranteed by Theorem D.3. Each of the copies will have an error parameter $\delta = \frac{1}{n \log(2\Delta)}$ and the $i$:th copy will have parameters $(c_i, R_i)$ with $c_i = c/2$ and $R_i = 2^{i-1}\text{MAXDIST}/(2\Delta)$. The operations now work as follows:

- Insert($p$): the point $p \in P$ is inserted into each of the $\log(2\Delta)$ copies of the gap data structure;

- Query($p$): we query the point $p$ in each of the $\log(2\Delta)$ copies and out of the returned points, we return the closest to $p$.

Since the gap data structure of Theorem D.3 is monotone we have that the resulting data structure satisfies monotonicity. That it succeeds with probability at least $1 - 1/n$ follows from the selection of $\delta$ and the union bound over $\log(2\Delta)$ many copies of the gap data structure. Furthermore the guarantees of the query operation (to find a nearby point) is satisfied: let $q$ be the closest point to $p$ and suppose that $2^{i-2} \cdot \text{MAXDIST}/(2\Delta) \leq \text{DIST}(p,q) \leq 2^{i-1} \cdot \text{MAXDIST}/(2\Delta)$. Then, on Query($p$), the $i$:th copy of the gap data structure is guaranteed to return a point within distance $c/2 \cdot 2^{i-1} \cdot \text{MAXDIST}/(2\Delta) = c \cdot 2^{i-2} \cdot \text{MAXDIST}/(2\Delta) \leq c \cdot \text{DIST}(p,q)$ of $p$. Finally the running time of the operations is $\log(2\Delta)$ times the running time of each operation in the gap data structure. Hence, since $\delta = 1/(n \log(2\Delta))$, the running time of the operations is $O\left(\log(2\Delta) \cdot d \cdot \left(n^2 \log(2\Delta)\right)^{O(1/c^2)}\right) = O\left(d \log(\Delta) \cdot (n \log(\Delta))^{O(1/c^2)}\right)$ as required.

## D.3 LSH Parameters in our Experiments

We use the locality sensitive hash families based on $p$-stable distribution introduced by Datar et al. [17]. We set the parameters as follows. We work with one scale, and we set the number of hash functions to be 15. Moreover, we set the collision parameter (referred to as $r$ in [17] to be 10).

# E REJECTIONSAMPLING Algorithm Analysis

In this section we analysis the REJECTIONSAMPLING algorithm. We start by proving approximation guarantee and then stating the main theorem.

## E.1 Analysis of Approximation Guarantee

In this section we prove that REJECTIONSAMPLING has an approximation guarantee of $O(c^6 \log(k))$. Hence, for a fixed $c$, it has the same asymptotic approximation guarantee as the standard implementation of K-MEANS++ but with the advantage that it runs in near-linear time. For simplicity we assume that the LSH data structure is successful throughout the whole analysis. That is, for any $p \in P$, Query($p$) returns a point within distance $c \cdot \delta$ where $\delta$ is the minimum distance from $p$ to a point inserted in the data structure.

Theorem 5.4 says that the probability to sample a center in REJECTIONSAMPLING is very close to the same $D^2$-distribution as in K-MEANS++. At first, it therefore appears rather intuitive that they should have the same approximation guarantee. However, the analysis of K-MEANS++ is rather sensitive to even small perturbations to the probability of sampling a center. Indeed, in a recent paper [11], it was proved that the version of K-MEANS++ where centers are sampled using an approximation of the $D^2$-distribution achieves an approximation guarantee of $O(\log(k)^2)$. To get a tight guarantee of $O(\log(k))$ was raised as an open problem. Our analysis does not resolve this question. Instead we use the additional *monotonicity* property of our LSH data structure (see Theorem 5.4) to circumvent the most technical difficulty of [11]. This allows us to establish the tight asymptotic approximation guarantee of our procedure. Similarly to the proof in [11], our analysis closely follows Dasgupta's

analysis of K-MEANS++ [16]. The main difference is a slight change of the "potential" function (see (1)). However, for the sake of completeness, we reproduce the complete analysis here.

**Notation:** Throughout the proof, we use the following notation. For a set $P' \subseteq P$ of the points and an (ordered) set of centers $S = \{s_1, \ldots, s_i\}$ let

- $\Phi(P', S)$ be the $k$-means cost of data points $P'$ with respect to the centers $S$, i.e.,

$$\Phi(P', S) = \sum_{x \in P'} \text{DIST}(x, S)^2 \,,$$

- $\Phi_{\text{LSH}}(P', S)$ be the $k$-means cost of data set $P'$ with respect to the centers $S$ when using the assignment given by the LSH data structure, i.e.,

$$\Phi_{\text{LSH}}(P', S) = \sum_{x \in P'} \text{DIST}(x, \text{Query}(x))^2 \,,$$

where the points of $S$ have been inserted into the data structure in the order $s_1, s_2, \ldots, s_i$.

Furthermore, we denote by $\text{OPT}_i(P)$ the cost of an optimal clustering of the data points $P$ using $i$ centers and we let $C_1^*, C_2^*, \ldots, C_k^*$ be the partition of $P$ into $k$ cluster in a fixed optimal solution (with $k$ centers).

### E.1.1 Two preliminary lemmas

We start our analysis with two preliminary lemmas which are very similar to lemmas in [11], which in turn are based on similar lemmas in the original K-MEANS++ paper [4].

As the first center is chosen uniformly at random in both REJECTIONSAMPLING and K-MEANS++, we can reuse the following statement from the original analysis.

**Lemma E.1 (Lemma 3.1 in [4])** *Let $s_1$ denote the first center chosen by* REJECTIONSAMPLING. *For each optimal cluster $C_i^*$,*

$$\mathbf{E}\left[\Phi(C_i^*, \{s_1\}) \mid s_1 \in C_i^*\right] \le 2 \cdot \text{OPT}_1(C_i^*) \,.$$

For the next lemma, we use that Theorem 5.4 says that a center $s$ is sampled with a probability in $[q/c^2, q \cdot c^2]$ where $q$ denotes the probability that $s$ would be sampled by the $D^2$-distribution. This allows us to use Lemma 5 in the noisy K-MEANS++ analysis:

**Lemma E.2 (Lemma 5 in [11])** *Consider* REJECTIONSAMPLING *after at least one center has been opened and let $S \ne \emptyset$ denote the current set of centers. We denote by $s$ the next sampled center. Then for any $S \ne 0$ and any optimal cluster $C_i^*$,*

$$\mathbf{E}\left[\Phi(C_i^*, S \cup \{s\}) \mid S, s \in C_i^*\right] \le 8c^4 \cdot \text{OPT}_1(C_i^*) \,.$$

### E.1.2 Dasgupta's potential argument

Consider a run of REJECTIONSAMPLING and let $S_i = \{s_1, s_2, \ldots, s_i\}$ denote the first $i$ centers chosen by REJECTIONSAMPLING (for notational convenience, we let $S_0 = \emptyset$). We say that a cluster $C_j^*$ of the optimal solution is covered by $S_i$ if one of its centers is in $C_j^*$. Otherwise we say that this cluster is uncovered. For $i \in \{0, 1, \ldots, k\}$, let $H_i$ and $U_i$ denote the set of all points from $P$ that, with respect to $S_i$, belong to covered and uncovered optimal clusters, respectively. Also let $u_i$ denote the number of uncovered clusters after $i$ centers were opened. Finally, we say that a center $s_i$ is wasted if $s_i \in H_{i-1}$, i.e., if the $i$:th center $s_i$ does not cover a previously uncovered cluster.

The following is an immediate corollary of the two preliminary lemmas; it is Corollary 6 in [11].

**Corollary E.3** *For any $i \in [k]$,*

$$\mathbf{E}\left[\Phi(H_i, S_i)\right] \le 8c^4 \cdot \text{OPT}_k(P) \,.$$

The above corollary, says that the cost of covered clusters is at most a constant times the cost of an optimal solution. To bound the expected cost of uncovered clusters we use the argument of [16]. It is based on a potential function argument. Define $W_i$ to be the number of wasted centers among the first $i$ centers. Hence $W_i$ equals $i$ minus the number of covered clusters. Further, let

$$\Psi_i = W_i \cdot \frac{\Phi_{\text{LSH}}(U_i, S_i)}{u_i} \,. \tag{1}$$

Our potential $\Psi_i$ is different from the one used in [16] in that we use $\Phi_{\text{LSH}}$ instead of $\Phi$. This is the main difference and it is crucial for our analysis.

For intuition, note that, for $i = 0$, we have no wasted centers and all clusters are uncovered. So $W_0 = 0$ and $u_0 = k$ and $\Psi_0 = 0$. At the other end (for $i = k$), we have that the number of wasted centers equals the number of uncovered clusters, i.e., $W_k = u_k$, and so $\Psi_k$ equals the total cost of uncovered clusters. The definition of $\Psi_i$ allows us to bound this cost step-by-step. In particular, we will bound the expected increase of (1) from $i$ to $i + 1$, i.e., $\mathbf{E}\left[\Psi_{i+1} - \Psi_i\right]$. We emphasize that the analysis is close to a verbatim transcript of that in [16]; it is included for completeness.

In the following, we let $\mathcal{F}_i$ denote the realization of REJECTIONSAMPLING of the first $i$ centers. Any realization $\mathcal{F}_i$ determines e.g. the values of $\Phi_{\text{LSH}}(U_i, S_i)$ and $u_i$.

We consider two cases: when the new center is in an uncovered cluster (Lemma E.4) and when it is in a covered cluster (Lemma E.5).

**Lemma E.4 (Lemma 8 in [16])** *Suppose that the $(i + 1)$:th center $s = s_{i+1}$ is chosen in $U_i$. Then for any $\mathcal{F}_i$*

$$\mathbf{E}\left[\Psi_{i+1} - \Psi_i \mid \mathcal{F}_i, \{s \in U_i\}\right] \leq 0 \,.$$

*Proof.* When $s$ belongs to an uncovered cluster $A$, we have $H_{i+1} = H_i \cup A$, $W_{i+1} = W_i$, $U_{i+1} = U_i \setminus A$, and $u_{i+1} = u_i - 1$. Hence, using the monotonicity of the LSH data structure,

$$\begin{aligned}
\Psi_{i+1} &= \frac{W_{i+1} \cdot \Phi_{\text{LSH}}(U_{i+1}, S_{i+1})}{u_{i+1}} \\
&\leq \frac{W_i \cdot (\Phi_{\text{LSH}}(U_i, S_i) - \Phi_{\text{LSH}}(A, S_i))}{u_i - 1} \,.
\end{aligned}$$

Let us bound the cost $\Phi_{\text{LSH}}(A, S_i)$ for a randomly chosen uncovered cluster $A$. Here we use the notation $A(s)$ to denote the uncovered cluster so that $s \in A$. Since a point $s \in U_i$ is sampled proportional to $\Phi_{\text{LSH}}(s, S_i)$

$$\begin{aligned}
\mathbf{E}&\left[\Phi_{\text{LSH}}(A(s), S_i) \mid \mathcal{F}_i, \{s \in U_i\}\right] \\
&= \sum_A \frac{\Phi_{\text{LSH}}(A, S_i)}{\Phi_{\text{LSH}}(U_i, S_i)} \cdot \Phi_{\text{LSH}}(A, S_i) \\
&\geq \frac{\Phi_{\text{LSH}}(U_i, S_i)}{u_i} \,,
\end{aligned}$$

where the sum is over the $u_i$ uncovered clusters $A$ and the last inequality is by the Cauchy-Schwarz inequality. Thus, $\mathbf{E}\left[\Psi_{i+1} \mid \mathcal{F}_i, \{s \in U_i\}\right]$ is at most

$$\begin{aligned}
\frac{W_i}{u_i - 1} &\left(\Phi_{\text{LSH}}(U_i, S_i) - \mathbf{E}\left[\Phi_{\text{LSH}}(A(s), S_i) \mid \mathcal{F}_i, \{s \in U_i\}\right]\right) \\
&\leq \frac{W_i}{u_i - 1}\left(\Phi_{\text{LSH}}(U_i, S_i) - \frac{\Phi_{\text{LSH}}(U_i, S_i)}{u_i}\right) = \Psi_i \,.
\end{aligned}$$

$\square$

**Lemma E.5 (Lemma 9 in [16])** *Suppose that $(i + 1)$ center $s = s_{i+1}$ is chosen in $H_i$. Then for any $\mathcal{F}_i$, $\Psi_{i+1} - \Psi_i \leq \Phi_{LSH}(U_i, S_i)/u_i$.*

*Proof.* When $s$ is chosen from a covered cluster, we have $H_{i+1} = H_i, U_{i+1} = U_i, u_{i+1} = u_i$, and $W_{i+1} = W_i + 1$. Thus by the monotonicity of our data structure

$$
\begin{aligned}
\Psi_{i+1} - \Psi_i &= \frac{W_{i+1} \cdot \Phi_{\text{LSH}}(U_{i+1}, S_{i+1})}{u_{i+1}} - \frac{W_i \cdot \Phi_{\text{LSH}}(U_i, S_i)}{u_i} \\
&\leq \frac{(W_i + 1) \cdot \Phi_{\text{LSH}}(U_i, S_i)}{u_i} - \frac{W_i \cdot \Phi_{\text{LSH}}(U_i, S_i)}{u_i} \\
&= \frac{\Phi_{\text{LSH}}(U_i, S_i)}{u_i} .
\end{aligned}
$$

$\square$

Putting these two lemmas together gives a bound on the expected increase of the potential.

**Lemma E.6 (Lemma 10 in [16])** *For $i \in \{0, 1, \ldots, k-1\}$ and $\mathcal{F}_i$, we have*

$$
\mathbf{E}\left[\Psi_{i+1} - \Psi_i \mid \mathcal{F}_i\right] \leq \frac{\Phi_{LSH}(H_i, S_i)}{k-i} .
$$

*Proof.* We have that $\mathbf{E}\left[\Psi_{i+1} - \Psi_i \mid \mathcal{F}_i\right]$ equals the sum of

$$
\mathbf{E}\left[\Psi_{i+1} - \Psi_i \mid \mathcal{F}_i, \{s \in U_i\}\right] \cdot \Pr[s \in U_i \mid \mathcal{F}_i]
$$

and

$$
\mathbf{E}\left[\Psi_{i+1} - \Psi_i \mid \mathcal{F}_i, \{s \in H_i\}\right] \cdot \Pr[s \in H_i \mid \mathcal{F}_i] .
$$

Now, by Lemma E.4 and Lemma E.5 together with the fact that REJECTIONSAMPLING samples a center in $H_i$ with probability $\frac{\Phi_{\text{LSH}}(H_i, S_i)}{\Phi_{\text{LSH}}(P, S_i)}$, we can upper bound this sum by

$$
0 + \frac{\Phi_{\text{LSH}}(U_i, S_i)}{u_i} \cdot \frac{\Phi_{\text{LSH}}(H_i, S_i)}{\Phi_{\text{LSH}}(P, S_i)} \leq \frac{\Phi_{\text{LSH}}(H_i, S_i)}{u_i} \leq \frac{\Phi_{\text{LSH}}(H_i, S_i)}{k-i} .
$$

$\square$

We are now ready to bound the overall cost of REJECTIONSAMPLING.

**Theorem E.7 (Theorem 11 in Dasgupta)** *If $S_k = S$ are the centers returned by* REJECTIONSAM-PLING *then*

$$
\mathbf{E}\left[\Phi(P, S)\right] \leq 8c^6 (\ln(k) + 2) \operatorname{OPT}_k(P) .
$$

*Proof.* Using $\Phi(P, S) = \Phi(H_k, S) + \Phi(U_k, S) \leq \Phi(H_k, S) + \Phi_{\text{LSH}}(U_k, S) = \Phi(H_k, S) + \Psi_k$, we have

$$
\begin{aligned}
\mathbf{E}\left[\Phi(P, S)\right] &\leq \mathbf{E}\left[\Phi(H_k, S)\right] + \sum_{i=0}^{k-1} \mathbf{E}\left[\Psi_{i+1} - \Psi_i\right] \\
&\leq \mathbf{E}\left[\Phi(H, S)\right] + \sum_{i=0}^{k-1} \mathbf{E}\left[\frac{\Phi_{\text{LSH}}(H_i, S_i)}{k-i}\right] \\
&\leq \mathbf{E}\left[\Phi(H, S)\right] + c^2 \sum_{i=0}^{k-1} \mathbf{E}\left[\frac{\Phi(H_i, S_i)}{k-i}\right] \\
&\leq 8c \cdot \operatorname{OPT}_k(P) + 8c^6 \sum_{i=0}^{k-1} \frac{\operatorname{OPT}_k(P)}{k-i} \\
&\leq 8c^6 (\ln(k) + 2) \operatorname{OPT}_k(P) ,
\end{aligned}
$$

where the second inequality is by Lemma E.6, the third inequality is by the assumption that the LSH data structure is successful and thus returns $c$-approximate distances, and the penultimate inequality is by Corollary E.3. $\square$

### E.2 Main Theorem for REJECTIONSAMPLING Algorithm

**Theorem 5.4** *For any constant $c > 1$, with probability at least $(1 - 1/n)$ REJECTIONSAMPLING always samples points $x$ that are at most a factor $c^2$ away from the $D^2$-distribution, its expected running time is $O\left(n \log(d\Delta)(d + \log n) + kc^2 d^3 \log(\Delta) \cdot (n \log(\Delta))^{O(1/c^2)}\right)$, and it returns a solution that in expectation is a $O(c^6 \log k)$-approximation of the optimal solution.*

*Proof.* With probability at least $1 - 1/n$, the LSH data structure is successful and we will show that the statements of the theorem holds if that is the case. We start by showing that REJECTIONSAMPLING samples points $x$ that are at most a factor $c^2$ away from the $D^2$-distribution. From Lemma 5.2 we know that the probability of sampling any point $x$ is $\frac{\text{DIST}(x,\text{Query}(x))^2}{\sum_{y \in P} \text{DIST}(y,\text{Query}(y))^2}$. Since (by the assumption that the data structure is successful) we have that $\text{DIST}(x, S) \leq \text{DIST}(x, \text{Query}(x)) \leq c \cdot \text{DIST}(x, S)$, so we have

$$\frac{\text{DIST}(x, S)^2}{c^2 \sum_{y \in P} \text{DIST}(y, S)^2} \leq \frac{\text{DIST}(x, \text{Query}(x))^2}{\sum_{y \in P} \text{DIST}(y, \text{Query}(y))^2}$$

$$\leq c^2 \cdot \frac{\text{DIST}(x, S)^2}{\sum_{y \in P} \text{DIST}(y, S)^2}.$$

The time to initialize the multi-tree embedding (MULTITREEINIT) is $O(nd \log(d\Delta))$, the time to initialize the data structure used by MULTITREEOPEN and MULTITREESAMPLE is $O(n \log(d\Delta))$, the total running time of MULTITREEOPEN is $O(n \log(d\Delta) \log n)$ (Lemma 4.1) and the running time of each call to MULTITREESAMPLE is $O(\log n)$ (Lemma 4.2). Finally, by Lemma 5.3, the expected number of iterations of the loop in REJECTIONSAMPLING is $O(c^2 d^2 k)$, and the running time of each iteration is dominated by the running time of the Insert and Query operations, which is $O\left(d \log(\Delta) \cdot (n \log(\Delta))^{O(1/c^2)}\right)$ by Theorem 5.1. Hence the total running time is $O\left(n \log(d\Delta)(d + \log n) + kc^2 d^3 \log(\Delta) \cdot (n \log(\Delta))^{O(1/c^2)}\right)$. The analysis of the approximation guarantee is presented in the previous section. $\square$

## F  Variance of the Experiments and Aspect Ratio

Table 7 and Table 8 presents the variance of the experiments. Recall that the numbers are reported over 5 runs.

The assumption of bounded aspect ratio allows a clean presentation of the result. The dependency can, for example, be removed (using ideas from prior works) if we have a very rough estimate of the optimum solution (e.g., within a factor $n$ or even $n^{10}$). Indeed, in that case, we can obtain an instance in which each coordinate of each point is an integer in range $[1, \text{poly}(n)]$ by losing a factor $1 + 1/n$ in the approximation guarantee (see [2]). This bounds $\log \Delta = O(\log(nd))$. In practice this can be achieved very efficiently. In order to bound the height of the tree, we propose the following:

- We first compute an estimate of optimum solution by sampling a solution of 20 randomly chosen points from the input. Then we compute the cost of this solution by assigning each point to the closest in the solution.

- Then we divide this value by number of point and number of coordinate and 200. This is intuitively the error that we let each coordinate make. The factor 200 is chosen to ensure that the total error made is within $0.5\%$ of the considered optimum value. The call this value the scaling factor.

- Afterwards, for each dimension of each point we divide it by the scaling factor and remove the fraction. For instance if the value of a considered coordinate is $1.2345$ and scaling factor is $0.01$, the resulting value would be $123$.

| Algorithm | $k = 100$ | $k = 500$ | $k = 1000$ | $k = 2000$ | $k = 3000$ | $k = 5000$ |
|---|---|---|---|---|---|---|
| FASTK-MEANS++ | 75364 | 169739 | 88843 | 92564 | 24225 | 40731 |
| REJECTIONSAMPLING | 288718 | 215658 | 74654 | 68922 | 87984 | 75364 |
| K-MEANS++ | 223686 | 64796 | 26784 | 20958 | 20881 | 30295 |
| AFKMC2 | 393782 | 121318 | 82700 | 22299 | 26945 | 15460 |
| UNIFORMSAMPLING | 687634 | 294580 | 147379 | 189350 | 182828 | 132779 |

Table 7: The variance of the solutions of the algorithms for the Song dataset for various values of $k$. All the numbers are scaled down by a factor $10^5$.

| Algorithm | $k = 100$ | $k = 500$ | $k = 1000$ | $k = 2000$ | $k = 3000$ | $k = 5000$ |
|---|---|---|---|---|---|---|
| FASTK-MEANS++ | 27110 | 672 | 813 | 86 | 77 | 163 |
| REJECTIONSAMPLING | 20440 | 1631 | 799 | 290 | 227 | 86 |
| K-MEANS++ | 8294 | 996 | 269 | 205 | 42 | 24 |
| AFKMC2 | 11529 | 830 | 883 | 204 | 495 | 135 |
| UNIFORMSAMPLING | 567214 | 290954 | 24118 | 23299 | 8770 | 23243 |

Table 8: The variance of the solutions of the algorithm for KDD-Cup dataset for various values of $k$. All the numbers are scaled down by a facto $10^2$.