A Study of Seeding Algorithms for K-Means Problem

Lazar Milenković

Supervisor: Ola Svensson

Master Semester Project

School of Computer and Communication Sciences
EPFL
June 14, 2018

Contents

1 Introduction

In this paper, we address the problem of analyzing the seeding techniques for *k-means* clustering problem. Clustering problems appear as a very natural in unsupervised learning, signal processing and analyzing datasets. Given a dataset, the goal of clustering algorithms is to divide data points into a number of groups by given similarity criteria. Depending on the similarity function, there are many different clustering problems. In this paper, we focus on *k-means* problem where the goal is to choose k centers such that the sum of the squared Euclidean distance from all dataset points to its nearest centers is minimized. As we will see in Section 2, the most widely used algorithm, Lloyd's algorithm, vastly depends on the way the initial points have been selected. The resulting clustering can have an arbitrary bad cost if the initial points were seeded improperly. The problem of seeding the initial points has been studied and many suggestions were proposed. A sedding technique that appeared in [1], *k-means++* algorithm, became standard due to its speed, simplicity and approximation ratio of $O(\log k)$. A natural question that arises is whether it is possible to come up with an algorithm which has the same running time and achieves constant factor approximation ratio.
The main contribution of this work is the algorithm given in Section 5, which achieves a constant factor approximation ratio and runs in $\tilde{O}(nk)$ time for one-dimensional datasets. Its high-dimensional adaptations are considered in 5.2 where possible techniques are suggested.
The rest of the paper is organized as follows. In Section 2, we describe the problems considered and notation used in the following sections. In Section 3, we give analysis of the competitive ratio of *k-means++* algorithm. Section 4 explains the online median algorithm, which is used as a starting point for the seeding algorithm defined in Section 5.

2 Preliminarities

In this section, we introduce the previous work that is useful for the rest of the report. First of all, we define a metric distance function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. In other words, we assume that the distance function is nonnegative, symmetric and satisfies triangle inequality, which is sometimes going to be weakened by λ-approximate triangle inequality.

Definition 1. We define λ-approximate triangle inequality ($\lambda \geq 1$) for any sequence of points $(x_0, \ldots, x_m)$ as follows:

$$d(x_0, \ldots, x_m) \leq \lambda \cdot \sum_{i=0}^{m-1} d(x_i, x_{i+1})$$

For a point y and arbitrary set of points X we also define $d(y, X) = \min_{x \in X} d(y, x)$. Finally we define $cost(X, Y) = \sum_{y \in Y} d(y, X)$. In the *k-means* problem, we are given a set of points (sometimes referred to as a dataset) $\mathcal{X} \subset \mathbb{R}^n$, integer k and a distance function $d(x, y)\|x - y\|_2^2$, which we will refer to as squared distance. We wish to choose k points from $\mathbb{R}^n$ and create clustering $\mathcal{C} = \{c_1, c_2, \ldots c_k\}$ such that $cost(\mathcal{C}, \mathcal{X}) = \sum_{x \in \mathcal{X}} d(x, \mathcal{C}) = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} \|x - c\|^2$ is minimized. For a point x and set Y we also define the following two notations $cost(x, Y) = \sum_{y \in Y} d(x, y)$ and $cost(Y, x) = \min_{y \in Y} d(y, x) = d(x, Y)$.

Definition 2 (*k-means* clustering)**.** Given a set of points $\mathcal{X}$ and a cost function $d : \mathbb{R}^2 \to \mathbb{R}$ (usually $d(x, y) = \|x - y\|^2$) we define *k-means* clustering as follows:

- *Input*: Finite set $\mathcal{X} \subset \mathbb{R}^d$; integer k.
- *Output*: $T \subset \mathbb{R}^d$ with $|T| = k$.

- *Goal*: Minimize $cost(T) = \sum_{x \in \mathcal{X}} \min_{z \in T} d(x, z)$.

Another importaint instance of clustering problem is *k-medians* where we are given a set of points $\mathcal{X} \subset \mathbb{R}^n$, integer k and a distance function $d(x, y) = \|x - y\|_1$. The goal is to find a subset of datapoints of cardinality k that minimizes the cost function $cost(\mathcal{C}, \mathcal{X}) = \sum_{x \in \mathcal{X}} d(x, \mathcal{C}) = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} \|x - c\|$. In other words, this problem differs from *k-means* in the distance function and in the condition that the points needs to be from the dataset.

Definition 3 (k-medians clustering)**.** Given a set of points $\mathcal{X}$ and a cost function $d : \mathbb{R}^2 \to \mathbb{R}$ we define k-median clustering as follows:

- *Input*: Finite set $\mathcal{X}$; integer k.
- *Output*: $T \subset \mathcal{X}$ with $|T| = k$.
- *Goal*: Minimize $cost(T) = \sum_{x \in \mathcal{X}} d(x, T)$.

In *online k-medians problem* we are given a dataset $\mathcal{X} \subset \mathbb{R}^n$ and we wish to define an ordering on $\mathcal{X}$. Competitive ratio of the ordering is given by maximum over all $0 < k \leq |\mathcal{X}|$ of the ratio of the cost of the configuration given by the first k points in the ordering to that of an optimal k-median configuration.

2.1 Lloyd's algorithm

One of the most widely used algorithm for *k-means* problem is Lloyd's algorithm. Although it is known to have an exponential run-time, it is widely used in practice due to its simplicity. The algorithm works by taking k uniformly random initial points and sets them as initial centers. Then it proceeds in stages. Each point from the dataset is assigned to its nearest center and the points are divided into clusters accordingly. Now, the center of each cluster becomes median of the points in the cluster. This is being repeated until an iteration doesn't change the cost. More formal definition of the algorithm follows.

1. Let $\mathcal{C} = \{c_1, c_2, \ldots, c_k\}$ be initial centers chosen uniformly at random from $\mathbb{R}^n$.
2. For each $x \in \mathcal{X}$ choose its closest center. Let $C_i = \{x \in \mathcal{X} \mid \arg\min_{c \in \mathcal{C}} d(x, \mathcal{C}) = c_i\}$ be the i-th cluster defined by these centers for each $i \in [k]$.
3. Let $c_i = \frac{1}{n} \sum_{x \in C_i} x = median(C_i)$ for each $i \in [k]$.
4. Repeat 2-3 until no longer cost changes occur.

It is easy to show that this algorithm always converges towards local optimum, depending on the initial set of points. More interesting is to notice that the performance of the algorithm can be arbitrarily bad, depending on the initial points chosen as the centers. Consider an example shown in Figure 1. We are given a dataset with four points x_1, x_2, x_3 and $x_4 \in \mathbb{R}^2$ and $k = 2$. Assume further that data points form a rectangle with the length of sides $2a$ and $2b$ and that initial points were chosen to be the centers of the two longer sides of the rectangle. In that case, Lloyd's algorithm converges immediately and the resulting cost is $4a^2$. The optimal solution would be to choose c_1 and $c2$ to be the centers of the shorter sides in which case the cost would be b^2. Choosing $a \gg b$ the cost becomes arbitrarily far from the optimal. This example shows that the way of sampling the initial points (seeding) determines the success of Lloyd's method. In the following sections, we will discuss potentially good seeding techniques and their approximation guarantees.

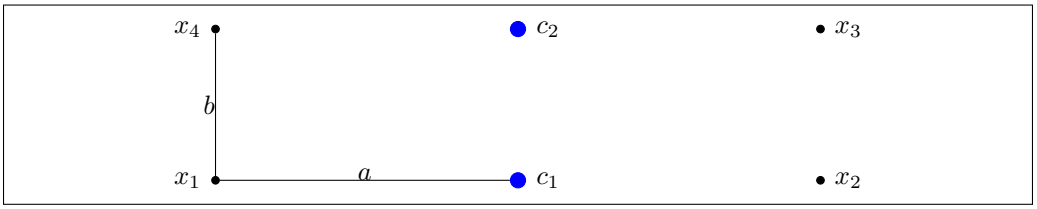

Figure 1: An example of bad initial seeding for Lloyd's algorithm. Consider $k = 2$ and the datapoints that form the rectangle. If we choose initial centers c_1 and $c2$ to be the centers of the longer sides of the rectangle, algorithm outputs an arbitrary bad solution if $a \gg b$.

3 k-means++

3.1 Algorithm description

In the previous section, we saw how a bad seeding can affect the performance of Lloyd's algorithm. This section is dedicated to the so-called *k-means++* algorithm, which is fairly simple to implement and is $O(\log k)$-competitive with the optimal clustering. It makes use of D^2 sampling where each center is sampled according to the squared distance from the current centers.

1. Choose an initial center c_1 uniformly at random from $\mathcal{X}$.
2. Choose the next center c_i, selecting $c_i = x' \in \mathcal{X}$ with probability $\frac{d(x')}{\sum_{x \in X} d(x)}$. Here, $d(x)$ defines the squared Euclidean distance of point x to the closest center choosen so far.
3. Repeat 2 until k centers are chosen.
4. Proceed as with the standard Lloyd's algorithm.

Intuitively, we sample the data points such that the points far away from the current set of centers are more probable to be chosen.

3.2 Analysis of the *k-means++* algorithm

The following theorem proves the optimality result for *k-means++*. We are going to prove it in the rest of the section.

Theorem 1. Given set of points $\mathcal{X}$, if clustering $\mathcal{C}$ is constructed with *k-means++*, then $cost(\mathcal{C}, \mathcal{X}) \leq 8(\ln k + 2) cost(\mathcal{C}^*, \mathcal{X})$, where $\mathcal{C}^*$ denotes the optimal clustering.

We will prove this theorem by first showing that the cost incurred by a single point chosen by D^2 weighing to a cluster is at most 8 times worse than the optimal one. Later, we will analyze what happens when the k points fail to choose optimal clusters. In order to proceed, we will need a few additional definitions and lemmas. For a given cluster C and its representative z we define $cost(C, z) = \sum_{x \in C} \|x - z\|_2^2$. This cost is minimized when $z = mean(C)$, i.e. when we pick a center of mass to be a center of a cluster. We will characterize the cost of a cluster when its center is not chosen to be its mean.

Lemma 1. For any set $C \subset \mathbb{R}^d$ and any $z \in \mathbb{R}^d$,

$$cost(z, C) = cost(mean(C), c) + |C| \cdot \|z - mean(C)\|_2^2.$$

Lemma 1 follows from a generic bias-variance decomposition of random vectors.

Lemma 2. Let $X \in \mathbb{R}^d$ be any random variable. For any $z \in \mathbb{R}^d$,

$$\mathbb{E}[\|X - z\|_2^2] = \mathbb{E}[\|X - \mathbb{E}[X]\|_2^2] + \|z - \mathbb{E}[X]\|_2^2$$

Proof. We expand the right-hand side,

$$\begin{aligned}
\mathbb{E}[\|X - \mathbb{E}[X]\|_2^2] + \|z - \mathbb{E}[X]\|_2^2 = \\
\mathbb{E}[\|X\|_2^2 + \|\mathbb{E}[X]\|_2^2 - 2X \cdot \mathbb{E}[X]] + (\|z\|_2^2 + \|\mathbb{E}[X]\|_2^2 - 2z \cdot \mathbb{E}[X]) = \\
\mathbb{E}[\|X\|_2^2] + \|\mathbb{E}[X]\|_2^2 - 2\mathbb{E}[X] \cdot \mathbb{E}[X] + \|z\|_2^2 + \|\mathbb{E}[X]\|_2^2 - 2z \cdot \mathbb{E}[X] = \\
\mathbb{E}[\|X\|]_2^2 + \|z\|_2^2 - 2z \cdot \mathbb{E}[X] = \mathbb{E}[\|X - z\|]_2^2.
\end{aligned}$$

□

Now we apply Lemma 2 to a uniformly random center from cluster C. We have

$$\mathbb{E}[\|X - z\|_2^2] = \sum_{x \in C} \frac{1}{|C|} \|x - z\|_2^2 = \frac{1}{|C|} cost(z, C)$$

and

$$\mathbb{E}[\|X - E[X]\|]_2^2 = \frac{1}{|C|} cost(mean(C), C)$$

which proves the Lemma 1.

Lemma 3. For any cluster $S \subset \mathcal{X}$ let μ_S be its center of mass. If we select center a uniformly at random from S then $\mathbb{E}[cost(a, S)] = 2cost(\mu_S, C)$.

Proof. We have by Lemma 1

$$\mathbb{E}[cost(a, S)] = \sum_{a \in S} \frac{1}{|S|} cost(a, S) = \frac{1}{|S|} \sum_{a \in S} (cost(\mu_S, S) + |S| \cdot \|a - \mu_S\|^2) = 2cost(\mu_S, S)$$

□

Lemma 4. Let A be an arbitrary cluster from $\mathcal{C}^*$ and let $\mathcal{C}$ be an arbitrary clustering. If we add a center $a \in A$ to $\mathcal{C}$ chosen with D^2 weighting, then $\mathbb{E}[cost(\mathcal{C} \cup \{a\}, A)] \leq 8cost(\mathcal{C}, A)$.

Proof.

$$\mathbb{E}[cost(\mathcal{C} \cup \{a\}, A)] = \sum_{a \in A} \frac{cost(\mathcal{C}, a)}{cost(\mathcal{C}, A)} \sum_{c \in A} \min(cost(\mathcal{C}, c), \|c - a\|^2))$$

Now, let x be an arbitrary point from A and let t be its closest center in $\mathcal{C}$. We have

$$cost(a, \mathcal{C}) \leq \|a - t\|^2 \leq (\|a - x\| + \|x - t\|)^2 \leq 2\|a - x\|^2 + 2cost(x, \mathcal{C}).$$

where we have used that $(u + v)^2 \leq 2u^2 + 2v^2$. Summing over all points from A, we obtain

$$cost(\mathcal{C}, a) \leq \frac{2}{|A|} \sum_{x \in A} \|a - x\|^2 + \frac{2}{|A|} \sum_{x \in A} cost(\mathcal{C}, x) = \frac{2}{|A|} cost(a, A) + \frac{2}{|A|} cost(\mathcal{C}, A).$$

Finally, we substitute this bound into earlier expression for the expected value

$$\sum_{a\in A}\frac{cost(\mathcal{C},a)}{cost(\mathcal{C},A)}\sum_{c\in A}\min(cost(\mathcal{C},c),\|c-a\|^2)) =$$
$$\frac{2}{|A|}\sum_{a\in A}\frac{cost(a,A)}{cost(\mathcal{C},A)}\sum_{c\in A}min(cost(\mathcal{C},c),\|c-a\|^2)$$
$$+\frac{2}{|A|}\sum_{a\in A}\frac{cost(A,\mathcal{C})}{cost(A,\mathcal{C})}\sum_{c\in A}min(cost(\mathcal{C},c),\|c-a\|^2)) =$$
$$\frac{4}{|A|}\sum_{a\in A}cost(a,A) = 8cost(\mu_A,A).$$

□

So this part finishes the proof in the case that we always chose points from the optimal clusters. Intuitively, Lemma 3.2 says that the uniformly random choice of the first cluster incurs the cost in expectation twice as big as the optimal one. Later, Lemma 4 guarantees that the cost in any other cluster is going to be at most 8 times bigger than the optimal. Rest of the analysis is devoted to the case when we failed to choose a center from the optimal cluster.
Let us define H_t to be the clusters of the optimal clustering that have been selected by *k-means++* by t-th iterations. In other words, these are the clusters of the optimal clustering such that *k-means++* chose at least one center from them. Let U_t be the set of clusters that are not covered by *k-means++* by t-th iteration. Also, we let w_t be the number of iterations in which we failed to choose an optimal cluster. It can be expressed as $w_t = t - |H_t|$. We will analyze the total cost by expression

$$cost_t(H_t) + \frac{w_t \cdot cost_t(U_t)}{|U_t|}.$$

Intuitively, the first expression $cost_t(H_t)$ is the cost of the covered clusters and by previous analysis we have that it is at most 8 times the optimal cost. In combination, these two expressions after choosing k clusters give us $cost(H_k)+cost(U_k) = cost(\mathcal{C})$ which is exactly the cost incurred by clustering C obtained by *k-means++*. We will focus on bounding the term

$$P_t = \frac{w_t \cdot cost_t(U_t)}{|U_t|}.$$

We will bound the expected increase in this amount from iteration t to $t+1$ by analyzing cases when the algorithm hits the covered and uncovered cluster.

Case 1 - Algorithm hits uncovered cluster A
In this case we have that $w_{t+1} = w_t$ and $|U_{t+1}| = |U_t| - 1$, so we have the following

$$P_{t+1} = \frac{w_{t+1}cost_{t+1}(U_{t+1})}{|U_{t+1}|} \leq \frac{w_t(cost_t(U_t) - cost_t(A))}{|U_t| - 1}.$$

We proceed by bounding the $cost_t(A)$ for randomly choosen A from U_t. The cost is bounded by:

$$\mathbb{E}[cost_t(A)] = \sum_{C\in U_t}\frac{cost(C)}{cost(U_t)}cost(C) \geq \frac{cost_t(U_t)}{|U_t|}$$

using the Cauchy-Schwarz inequality. Thus

$$\mathbb{E}[P_{t+1}] \leq \frac{w_t}{|U_t|-1}(cost_t(U_t) - \frac{cost_t(U_t)}{|U_t|}) = \frac{w_t}{|U_t|-1} cost_t(U_t) \frac{|U_t|-1}{|U_t|} = P_t,$$

from where we have that $\mathbb{E}[P_{t+1} - P_t] \leq 0$.

Case 2 - Algorithm hits already covered cluster A
In this case we have that $H_{t+1} = H_t$, $w_{t+1} = w_t + 1$ and $U_{t+1} = U_t$, so we obtain

$$P_{t+1} - P_t = \frac{w_{t+1} cost_{t+1}(U_{t+1})}{|U_{t+1}|} - \frac{w_t cost_t(U_t)}{|U_t|} = \frac{(w_t+1) cost_{t+1}(U_t)}{|U_t|} - \frac{w_t cost_t(U_t)}{|U_t|} \leq \frac{(w_t+1) cost_t(U_t) - w_t cost(U_t)}{|U_t|} = \frac{cost_t(U_t)}{|U_t|}.$$

Putting both cases together we obtain the following lemma.

Lemma 5. For any $t \geq 0$ we have $\mathbb{E}[P_{t+1} - E[P_t]] \leq cost_t(H_t)/(k-t)$.

Proof. In order to upper bound the expectation, we will consider only the second case (since it gives a nonnegative value) which occurs with probability $cost_t(H_t)/cost(\mathcal{C})$ by the definition of *k-means++* algorithm.

$$\mathbb{E}[P_{t+1} - P_t] \leq \frac{cost_t(U_t)}{|U_t|} \frac{cost_t(H_t)}{cost(\mathcal{C}_t)} \leq \frac{cost(H_t)}{|U_t|} \leq \frac{cost(H_t)}{k-t}.$$

□

Finally, Theorem 1 follows by Lemmas 4 and 5:

$$\mathbb{E}[cost(\mathcal{C})] = \mathbb{E}[cost_k(H_k)] + \sum_{t=0}^{k-1} \mathbb{E}[P_{t+1} - P_t] \leq \mathbb{E}[cost_k(H_k)] + \sum_{t=0}^{k-1} \frac{cost_t(H_t)}{k-t} \leq 8cost(\mathcal{C}^*)(1 + \sum_{t=0}^{k-1} \frac{1}{k-t}) \leq 8cost(\mathcal{C}^*)(2 + lnk).$$

Authors also show in [1] that the expected cost incurred by the *k-means++* is tight by giving a high-dimensional example. Further work by [4] gives a simple two dimensional dataset where the *k-means++* achieves a tight approximation ratio.

4 Algorithm for the online median problem

4.1 Algorithm description

In this section, we describe an algorithm for online median problem given in [2]. Original algorithm was more general and introduced the weight function assigned to each node. We will ignore the weight and focus only on the cost as defined in the online median poblem. First we define useful constants:

$$\lambda \geq 1$$

$$\alpha > 1 + \lambda$$

$$\beta \geq \frac{\lambda(\alpha - 1)}{\alpha - 1 - \lambda}$$

$$\gamma \geq (\frac{\alpha^2\beta + \alpha\beta}{\alpha - 1} + \alpha)\lambda$$

In order to define the algorithm we will need a few additional definitions:

- A *ball* $A = (x, r)$ is defined with its center x and radius $r \in \mathcal{X}$.
- Given a ball $A = (x, r)$ we define $points(A) = \{y \in \mathcal{X} : d(x, y) \leq r\}$.
- The *value* of a ball $A = (x, r)$ is defined as $\sum_{y \in A}(r - d(x, y))$.
- Given a ball $A = (x, r)$ we define its scaled version $cA = (x, cr)$.
- A *child* of (x, r) is any ball $(y, \frac{r}{\alpha})$, where $d(x, y) \leq \beta r$.
- For any nonempty sequence ρ, we let $head(\rho)$ and $tail(\rho)$ denote the first and the last element of ρ, respectively.
- Finally, let $isolated(x, \mathcal{C})$ be a ball $(x, d(x, \mathcal{C})/\gamma)$ and $isolated(x, \emptyset) = \max_{y \in \mathcal{X}} d(x, y)$.

Our goal is to define a sequence of clusterings $\mathcal{C}_0, \mathcal{C}_1, \ldots \mathcal{C}_n$ so that $\max_{i \in [n]} cost(\mathcal{C}_i)/cost(\mathcal{C}_i^*)$ is minimized.
The algorithm is defined as follows:

1. Let $\mathcal{C}_0 = \emptyset$ and for $i = 0$ to $n - 1$ execute the following steps.
2. Let σ_i denote the signleton sequence $\langle A \rangle$ where A is a maximum value ball in $\{isolated(x, \mathcal{C}_i) : x \in \mathcal{X} \setminus \mathcal{C}_i\}$
3. While the ball $tail(\sigma_i)$ has more than one child, append a maximum value child of $tail(\sigma_i)$ to σ_i.
4. Set $\mathcal{C}_{i+1} = \mathcal{C}_i \cup \{center(tail(\sigma_i))\}$.

4.2 Analysis of the competitive ratio

In this part, we show that the online median algorithm achieves the constant competitive ratio. In order to proceed with the analysis, we will need several more definitions.

- A $configuration$ is any nonempty subset of dataset $\mathcal{X}$
- Let z_i be the unique point in $Z_{i+1} \setminus Z_i$, $0 \leq i < n$.
- For any configuration X, we divide $\mathcal{X}$ into $|X|$ sets $\{cell(x, X) \mid x \in X\}$ as follows: for each point y in $\mathcal{X}$, we choose a point x in X such that $d(y, X) = d(y, x)$ and add y to $cell(x, X)$.
- For any configuration X, point x in X, and set of points Y, we define $in(x, X, Y)$ as $cell(x, X) \cap isolated(x, Y)$ and $out(x, X, Y)$ as $cell(x, X) \setminus in(X, X, Y)$.

- For any configuration x and set of points Y, we let $in(X, Y)$ denote the set $\cup_{x \in X} in(x, X, Y)$ and $out(X, Y) = \mathcal{X} \setminus in(X, Y)$.

Throughout the rest of the analysis, we will usually consider sets $in(x, X, Z_{|X|})$ and $out(x, X, Z_{|X|})$. Furthermore, we will split our analysis into two parts. First, we show that for any point $y \in out(X, Z_{|X|})$, distance $d(y, Z_{|X|})$ is within the constant factor of $d(y, X)$. Later, we show that the cost incurred by $Z_{|X|}$ to the set $in(x, X, Z_{|X|})$ is within the constant factor of optimal. In order to minimize the competitive ratio of $2\lambda(\gamma + 1)$, we set $\lambda = 1$, $\alpha = 2 + \sqrt{3}$ and set β and γ to the right-hand sides of the inequalities defining them. We begin by stating the main theorem which describes the competitive ratio of the algorithm.

Theorem 2. For any configuration X, $cost(Z_{|X|}) \leq 2\lambda(\gamma + 1) \cdot cost(X)$

Proof. Let $Y = in(X, Z_{|X|})$, and let $Y' = out(X, Z_{|X|})$. Noting that $cost(X) = cost(X, Y) + cost(X, Y')$ and $cost(Z_{|X|}) = cost(Z_{|X|}, Y) + cost(Z_{|X|}, Y')$ the theorem follows from lemmas 7, 9 and 10. □

Thus we have splitted our cost into the inner and outer part. The following lemmas show that the cost of outer points are within the competitive ratio, whereas the Lemma 10 bounds the cost of the inner part.

Lemma 6. For any configuration X, and points x in X and y in $out(x, X, Z_{|X|})$, $d(y, Z_{|X|}) \leq \lambda(\gamma + 1) \cdot d(y, X)$.

Proof. Let $isolated(x, Z_{|X|}) = (x, r)$. It follows by the definition of *isolated* that $d(x, y) > r$ and that there exists a point $z \in Z_{|X|}$ such that $d(x, z) = \gamma r$. Hence, $d(y, Z_{|X|}) \leq d(y, z) \leq \lambda(d(x, y) + d(y, X)) = \lambda(d(x, y) + \gamma r) < \lambda(d(x, y) + \gamma \cdot d(x, y)) = \lambda(\gamma + 1)d(x, y) = \lambda(\gamma + 1)d(y, X)$. □

Thus we have shown that the cost incurred by a point far away from centers of an arbitrary clustering X is within a competitive ratio. The following lemma extends the result.

Lemma 7. For any configuration X, $cost(Z_{|X|}, out(X, Z_{|X|})) \leq \lambda(\gamma + 1) \cdot cost(X, Z_{|X|})$.

Proof. Summing the inequality of Lemma 6 over all y in $out(x, X, Z_{|X|})$, we obtain

$$cost(Z_{|X|}, out(x, X, Z_{|X|})) \leq \lambda(\gamma + 1) \cdot cost(X, out(x, X, Z_{|X|})).$$

The result follows by summing the above inequality over all x in X. □

Now we switch our attention to the inner part of the configuration X, i.e. to the points that are closer to a center in X. In a similar fashion as before, we analyze the case of a single point (Lemma 8) and generalize to the whole outer part in Lemma 9 by simply summing the terms.

Lemma 8. For any configuration X and point $x \in X$,

$$cost(Z_{|X|}, in(x, X, Z_{|X|})) \leq \lambda(\gamma + 1)(cost(X, in(x, X, Z_{|X|})) + value(isolated(x, Z_{|X|}))).$$

Proof. Let $isolated(x, Z_{|X|}) = (x, r)$. Note that $d(x, y) = \gamma r$ for some $y \in Z_{|X|}$. Thus, for any z in $isolated(x, Z_{|X|})$, $d(y, z) \leq \lambda(d(y, x) + d(x, z) \leq \lambda(\gamma + 1)r$. It follows that

$$cost(Z_{|X|}, in(x, X, Z_{|X|})) \leq \lambda(\gamma + 1) \sum_{z \in in(x, X, Z_{|X|})} r \leq$$

$$\lambda(\gamma + 1)(\sum_{z \in in(x, X, Z_{|X|})} d(x, z) + \sum_{z \in isolated(x, Z_{|X|})} (r - d(x, z))) =$$

$$\lambda(\gamma + 1)(cost(X, in(x, X, Z_{|X|})) + value(isolated(x, Z_{|X|}))).$$

□

Lemma 9. For any configuration X and point x in X,

$$cost(Z_{|X|}, in(X, Z_{|X|})) \leq \lambda(\gamma + 1)(cost(X, in(X, Z_{|X|})) + value(isolated(x, Z_{|X|})).$$

Proof. The claim follows by summing the inequality of Lemma 8 over all $x \in X$. □

The only unbounded term left is $value(isolated(x, Z_{|X|}))$. Now we state the main technical lemma and spend the rest of te section proving it.

Lemma 10. For any configuration X,

$$\sum_{x \in X} value(isolated(x, Z_{|X|})) \leq cost(X).$$

In order to prove this claim, we will introduce a prunning procedure that will be applied to lists $sigma_i$ defined in the online median algorithm. The definition of the prunning procedure will require some further analysis of the lists σ_i. First, we make sure that any ball in σ_i is at least γr away from the current clustering produced by the online median algorithm.

Lemma 11. Let $A = (x, r)$ belong to σ_i. Then $d(x, Z_i) \geq \gamma r$.

Proof. Let z be a point in Z_i such that $d(x, z) = d(x, Z_i)$. If $A = head(\sigma_i)$ then the result is immediate. Otherwise, let $B = (y, s)$ denote the predcessor of A in σ_i, and assume inductively that $d(y, Z_i) \geq \gamma s$. Noting that $d(x, y) \leq \beta s$ and $s = \alpha r$ it follows,

$$d(x, Z_i) = d(x, z) \geq \frac{d(y, z)}{\lambda} - d(x, y) \geq (\frac{\gamma}{\lambda} - \beta)\alpha r \geq \gamma r$$

where the last inequality is obtained by the definition of γ. □

The following lemma is essential to soundness of the definition of the prunning procedure.

Lemma 12. Let $A = (x, r)$ belong to σ_i and let $B = (y, s)$ belong to σ_j. If $i < j$ and $d(x, y) \leq r + s$, then the following holds:

(i) $radius(head(\sigma_j)) \leq \frac{r}{\alpha}$

(ii) $A \neq tail(\sigma_i)$

(iii) Let C be the successor of A in σ_i. Then $value(C) \geq value(head(\sigma_j))$

Proof. Let $head(\sigma_j) = (y', s')$. We prove first part by deriving upper and lower bounds on $d(y', z_i)$. For a lower bound note that $d(y', z_i) \geq d(y', Z_j)$, since $i < j$ and $d(y', Z_j) \geq \gamma s'$ by Lemma 11. To derive an upper bound, we first let P denote the prefix of sequence σ_j, ending with ball B and we let S denote the suffix of sequence σ_i beggining with ball A. We then apply the λ-approximate triangle inequality to the sequence of points $\langle y', \dots y, x, \dots z_i \langle$, where the prefix $\langle y', \dots y\rangle$ corresponds tto the centers of the balls in P and the suffix $\langle x, \dots z_i\rangle$ corresponds to the centers of the balls in S. By repeated application of the definition of a child, and using the given upper bound on $d(x, y)$, we obtain:

$$d(y', z_i) \leq \lambda(\beta(s' + \frac{s'}{\alpha} + \dots + \alpha s) + s + r + \beta(r + \frac{r}{\alpha} + \dots)) \leq$$
$$(\frac{\alpha\beta}{\alpha - 1} \cdot (r + s') + r)\lambda.$$

Combining the bounds on $d(y', z_i)$ and applying the definition of γ, we obtain

$$(\frac{\alpha^2\beta}{\alpha-1}+\alpha)\lambda s' \leq (\frac{\alpha\beta}{\alpha-1}(r+s')+r)\lambda.$$

Multiplying through by $(\alpha-1)/\lambda$ and rearranging, we get $r \geq \frac{\alpha^2\beta+\alpha^2-\alpha}{\alpha\beta+\alpha-1} \cdot s' = \alpha s'$.
For the second part, note that $d(x,y) \leq r + \frac{r}{\alpha} < \beta r$ by the previous part and the definition of β. Thus A has at least two children and the claim follows.
Finally, for the third part we obtain an upper bound on $d(x,y')$ by applying the λ-approximate triangle inequality to the sequence of points $\langle y', \ldots, y, x\rangle$, where the prefix $\langle y', \ldots, y$ corresponds to the centers of the balls in P. We observe that

$$d(x,y') \leq \lambda(r+s+(\alpha s+\alpha^2 s+\cdots+s')\beta).$$

Then, by using the definitions of β and γ and part (i),

$$\lambda(r+s+(\alpha s+\alpha^2 s+\cdots+s')\beta) \leq$$
$$\lambda r+\frac{\alpha\beta\lambda}{\alpha-1}s' \leq \lambda r+\frac{\alpha\beta\lambda}{\alpha-1}\cdot\frac{r}{\alpha} \leq (\frac{\beta}{\alpha-1}+1)\lambda r \leq \beta r$$

It then follows that $head(\sigma_j)$ is contained in a child of A. Thus $value(C) \geq value(head(\sigma_j))$. □

In the remining part of the analysis, we fix a configuration X, and let $k = |X|$. We now define a *prunning procedure* that we use for the purpose of the analysis. The pruning procedure takes as input the k sequences σ_i, $0 \leq i < k$ and produces as output k sequences τ_i, $0 \leq i < k$. The pruning procedure then performs a number of iterations. In each of the iterations, the procedure checks if there exists two different balls $A = (x, r)$ and $B = (y, s)$ in distinct sequences τ_i and τ_j, respectively, such that $i < j$ and $d(x, y) \leq r + s$. If so, the sequenc τ_i is redefined as the proper suffix of the current τ_i beginning at the successor of A. If there are no such two balls, the procedure terminates. Note that part (ii) of the Lemma 12 ensures that this procedure is well defined. Also, since the procedure reduces the length of some sequence τ_i, it is guaranteed to terminate. Having defined the procedure, we examine some useful properties of the outputted lists τ_i.

Lemma 13. Let $A = (x, r)$ belong to τ_i and let $B = (y, s)$ belong to τ_j. If $i < j$ then $d(x, y) > r + s$.

Proof. The result is immediate from the definition of the pruning procedure. □

Lemma 14. Each sequence τ_i is nonempty.

Proof. The result is immediate from part (ii) of Lemma 12 and the definition of the pruning procedure. □

We proceed with proving few trivial results stated in lemmas 15 and 16 which connect value of head of some list τ_i and value of an isolated ball at step k. They lead to Lemma 17 which will later be useful to upper bound the cost that a distant point incures to our clustering.

Lemma 15. Let x be a point, and assume that $0 \leq i < j < n$. Then

$$value(isolated(x, Z_i)) \geq value(isolated(x, Z_j)).$$

Proof. Since $Z_i \subseteq Z_j$, $radius(isolated(x, Z_i)) \geq radius(isolated(x, Z_j))$ and the claim follows. □

Lemma 16. Let x be a point, and assume that $0 \leq i < k$. Then

$$value(head(\sigma_i)) \geq value(isolated(x, Z_k)).$$

Proof. If $x \in Z_i$, then $radius(isolated(x, Z_i)) = 0$ and there is nothing to prove. Otherwise, $value(head(\sigma_i)) \geq value(isolated(x, Z_i))$ by the definition of the online median algorithm and the claim follows by Lemma 15. □

Lemma 17. Let x be a point and assume that $0 \leq i < k$. Then

$$value(head(\tau_i)) \geq value(isolated(x, Z_k)).$$

Proof. We prove that the claim follows before and after each step of the pruning procedure. Initially, $\tau_i = \sigma_i$ and the result holds by Lemma 16. If it holds before an iteration of the pruning procedure, then it also holds after by part (iii) of Lemma 12. □

Finally, we will classify the balls according to the distance from the clustering X. We say that a ball $A = (x, r)$ is *covered* iff $d(x, X) < r$. A ball is uncovered iff it is not covered. Before proceeding with the final part of the proof, let us bound a value of a ball in terms of the cost that in incurs to the clustering X.

Lemma 18. For any uncovered ball $A = (x, r)$, $cost(X, A) \geq value(A)$.

Proof. Note that $cost(X, A) \geq \sum_{y \in A} d(y, X) \geq \sum_{y \in A}(r - d(y, x)) = value(A)$. □

Let I denote the set of all indices $i \in [k]$ such that some ball in τ_i is covered. We now construct a matching between the sets $[k]$ and X as follows. First, for each i in I we match i with a point x in X that belongs to the last covered ball in the sequence τ_i. Such point x always exists by the definition of I. Furthermore, Lemma 13 guarantees that we do not match the same point with more than one index. Second, for each point $i \in [k] \setminus I$, we match i with an arbitrary unmached point x in X. We construct a function ϕ that maps each point x in X to an uncovered ball. We set $\phi(x)$ to $head(\tau_i)$ for each i in $[k] \setminus I$. For each x in X that is matched with an index i in I, we set $\phi(x)$ to the successor of the last covered ball in τ_i, unless $tail(\tau_i)$ is covered, in which case we set $\phi(x)$ to the ball $(x, 0)$.

Lemma 19. For any pair of distinct points x and y in X, $\phi(x) \cap \phi(y) = \emptyset$.

Proof. The proof is immediate from 13 and the fact that the ball $(x, 0)$ is contained in $tail(\tau_i)$. □

Lemma 20. For any point x in X, $value(\phi(x)) \geq value(isolated(x, Z_k))$.

Proof. If x is matched with an index i in $[k] \setminus I$, the claim follows by Lemma 17. If x is matched with an index i from I, we consider two cases. If $tail(\tau_i)$ is covered, then $x = z_i$, since $tail(\tau_i)$ has exactly one child. The claim follows since $\phi(x) = isolated(x, Z_k) = (x, 0)$. If $tail(\tau_i)$ is uncovered, then the predcessor of $\phi(x)$ in τ_i, $A = (y, r)$ exists and contains x. It follows that $value(\phi(x)) \geq value(B)$, where $B = (x, \frac{r}{\alpha})$ is the child of A centered at x. Let $C = (x, s)$ denote the ball $isolated(x, Z_k)$. We complete the proof of the claim by showing that $\frac{r}{\alpha} \geq s$, which implies that $B \supseteq C$ and hence $value(B) \geq value(C)$. We prove this by deriving upper and lower bounds on $d(x, z_i)$. Let S be the suffix of the sequence τ_i beggining with the ball A. For the upper bound, we apply the triangle inequality to the sequence of points $\langle x, y, \ldots, z_i \rangle$, where the suffix $\langle y, \ldots, z_i \rangle$ consists of the centeres of the balls in S. We then obtain

$$d(x, z_i) \leq \lambda(r + \beta(r + \frac{r}{\alpha} + \cdots)) \leq (1 + \frac{\alpha\beta}{\alpha - 1})\lambda r$$

The desired inequality follows since $d(x, z_i) \geq \gamma s$ by the definition of C. □

Lemmas 18, 19 and 20 together yield a proof of Lemma 10.

4.3 Analysis of the run-time complexity

To analyze the running time of the algorithm, the following lemma will be useful.

Lemma 21. Let $A = (x, r)$ be a child of a ball B in sequence σ_i, and let $A' = (x, r')$ be a child of a ball B' in sequence σ_j. If $i < j$ then $r \geq (\alpha + 1 + \frac{1}{\beta})r'$.

Proof. We obtain the upper bound on $d(x, z_i)$ using the λ-approximate triangle inequality on a sequence of points consisting of the centers of the balls in the suffix of σ_i, begiiging with ball A.

$$d(x, z_i) \leq \lambda\beta(r + \frac{r}{\alpha} + \cdots) \leq \lambda\alpha\beta r/(\alpha - 1).$$

By Lemma 11 and since $j > i$, we get that $\gamma r' \leq d(x, Z_j) \leq d(x, z_i)$. Combinint these inequalities and using the definition of γ we obtain

$$r \geq \frac{(\alpha - 1)\gamma}{\lambda\alpha\beta} \cdot r' \geq \frac{\alpha - 1}{\alpha\beta} \cdot (\frac{\alpha^2\beta}{\alpha - 1} + \alpha)\lambda \cdot r' = (\alpha + 1 + \frac{1}{\beta})r'.$$

□

We will discuss two different implementation of the algorithm. The quantity used to express the runnig time is $l = \log \frac{\Delta}{\delta}$ where Δ and δ stands for the largest and smallest distance between two dataset points, respectively. In the first one, for each point $x \in \mathcal{X}$, we sort the remaining points according to distance to x in time $O(n^2 \log n)$. Time required to peform the first step of each iteration is $O(n)$ since we can keep track of distance of each point to the nearest center in the current cluster. Due to Lemma 21 we have that the total running time of the second step over all iterations is $O(n\, l \log n)$. This follows because each ball apears at most l times as a child of some other ball and the time required to determine the value of a ball is $O(\log n)$ by using a binary search. Overall the running time of this implementation is $O((n + l) \log n)$.
The second implementation supposes that we round down all distances to the nearest integral part of λ. Then for each point we can construct a $O(l)$-sized table of distances which can be used to determine the value of each ball in a constant time. This preprocessing can be done in $O(n + l)$ for each point. The second step of algorithm now becomes $O(n\, l)$ so the total running time is $O((n + l) \cdot n)$.

5 An algorithm for the *k-means* problem in lower dimensions

In this section, we will introduce some modifications of the algorithm for the online median problem in order to keep constant-factor competitive ratio while improving the running time of the algorithm. The first part explains the algorithm for one-dimensional Euclidean space, whereas the second part suggests its modifications for higher dimensions. As it was stated in section 4.1, we can use the algorithm for the online median problem even with the squared ℓ_2 norm (used in the *k-means* problem) and obtain the constant competitive ratio.

5.1 Algorithm for one-dimensional datasets

Restricting ourselves to one-dimensional case gives a natural relaxation to the problem since we can find a geometric ordering of the points by sorting them. In this case, it turns out that one can improve the running time of the algorithm to $O(n\, k\, l \log n)$. The quadratic cost in n for the online

median problem was due to creating vectors of points sorted by distance for each of the dataset points. In one dimensional case, this can be acieved by only one sorting at the beggining of the algorithm. This results in $O(n \log n)$ iterations in the preprocessing phase. Second observation used is that we can obtain the values of balls in $O(\log n)$ time using linear preprocessing after sorting the points. To further clarify, we rewrite the value of a ball $A = (x, r)$ as follows:

$$value(A) = \sum_{y \in A} r - d(x, y) = |A| \cdot r - \sum_{\substack{y \in \mathcal{X} \\ x-r \leq y \leq x+r}} d(x, y) =$$

$$|A| \cdot r - \sum_{\substack{y \in \mathcal{X} \\ x-r \leq y \leq x+r}} (x - y)^2 =$$

$$|A| \cdot r - |A| \cdot x^2 - (\sum_{\substack{y \in \mathcal{X} \\ x-r \leq y \leq x+r}} y^2) + (2x \sum_{\substack{y \in \mathcal{X} \\ x-r \leq y \leq x+r}} y)$$

Since we have a sorted array of points, we can find indices l and r such that $x - r \leq p_l$ and $p_r \leq x + r$, where p is the sorted array of dataset points. Thus the equation for calculating the values becomes

$$|A| \cdot r - |A| \cdot x^2 - (\sum_{l \leq i \leq r} p_i^2) + 2x(\sum_{l \leq i \leq r} p_i)$$

So we reduced the problem of finding the value of a ball to performing the range queries on the sorted array of dataset points. Note that our array of input points is not modified, so we can define the following sums:

$$s_i = 0, \text{ for } i < 0$$
$$s_0 = p_0$$
$$s_i = s_{i-1} + p_i, \text{ for } i > 0$$

and

$$t_i = 0, \text{ for } i < 0$$
$$t_0 = p_0$$
$$t_i = t_{i-1} + p_i^2, \text{ for } i > 0$$

Calculating of the arrays s and t takes place after the points are sorted, and requires $O(n)$ time and space. This time is dominated by sorting the array so we do not consider it in the further analysis. We have that $\sum_{l \leq i \leq r} p_i = s_r - s_{l-1}$, so the value query obtains the following form:

$$|A| \cdot r - |A| \cdot x^2 - (t_r - t_{l-1}) + 2x(s_r - s_{l-1}).$$

Final part of the algorithm to consider is traversing all the children of a ball. Recall that a child of a ball (x, r) is defined to be any ball $(y, \frac{r}{\alpha})$ where $d(x, y) \leq \beta r$. Imagine we have a ball with center x where $x = p_i$ for some index i of the sorted array of points. Then, we can increase i until $d(x, p_i)$ becomes bigger than βr and obtain the values of any ball on the way. Similarly for the rignt sand of point x, we increase i unti $d(x, p_i)$ becomes bigger than βr. Final version of the algorithm is given below.

Algorithm 1 One dimensional *k-means* seeding algorithm

```
1: function OneDimensionalSeed(x, k)          ▷ x: array of input points ;
2:                                            ▷ k: the number of centers to be placed;
3:     p ← sorted(x)
4:     s_0 ← p_0
5:     for i ← 0 to n − 1 do
6:         s_i ← s_{i−1} + p_i
7:     t_0 ← p_0^2
8:     for i ← 0 to n − 1 do
9:         t_i ← t_{i−1} + p_i^2
10:    for i ← 0 to n − 1 do
11:        dist(i) = max(d(p_i, p_0), d(p_i, p_{n−1}))
12:    for i ← 0 to k − 1 do
13:        best_value ← 0
14:        idx ← 0
15:        r ← 0
16:        for j ← 0 to n − 1 do
17:            if best_value < GetValue(i, dist(i)/γ) then
18:                best_value ← GetValue(i, dist(i)/γ)
19:                r ← dist(i)/γ
20:                idx ← i
21:        children ← 2
22:        while children > 1 do
23:            children ← 0
24:            best_idx ← idx
25:            best_value ← GetValue(i, r)
26:            i ← idx
27:            while d(i, idx) < βr do
28:                children ← children + 1
29:                if best_value < GetValue(i, r/α) then
30:                    best_value ← GetValue(i, r/α)
31:                    best_idx ← i
32:                i ← i − 1
33:            i ← idx + 1
34:            while d(p_i, p_idx) < βr do
35:                children ← children + 1
36:                if best_value < GetValue(i, r/α) then
37:                    best_value ← GetValue(i, r/α)
38:                    best_idx ← i
39:                i ← i + 1
40:            idx ← best_idx
41:            r ← r/α
42:        for i ← 0 to n do
43:            dist(i) ← min(dist(i), d(p_i, p_idx))
```

In the preprocessing phase (lines 3-11), we sort the points, calculate arrays s and t and for each point calculate the squared Euclidean distance to the farthest point in the dataset. Calculating

farthest point is needed for the first step of the algorithm when the $isolated(x, \emptyset)$ is defined as the ball $(x, \max_{y \in \mathcal{X}} d(x, y))$. Since the points lay on the on the line, the most distant point for each of the points either the leftmost or the rightmost point of the dataset. Overall the preprocessing phase is dominated by sorting which has a running time of $O(n \log n)$. In the lines 13-20, we find the point with the biggest value and it takes $O(n \log n)$ time, where $\log n$ factor comes from calling the GetValue procedure. Finally, lines 22-41 traverses the children of the highest value ball until it reaches a ball with only one child (itself). By the Lemma 21 we have that these steps take $nkl \log n$ time in total where $l = \log \frac{\Delta}{\delta}$, where Δ and δ are the biggest and smallest squared distances between two dataset points, respectively. Factor $\log n$ comes again due to the binary search in GetValue. Thus, we have achieved the overall running time of $O(n\,k\,l\,\log n)$.

5.2 Algorithm for $\mathbb{R}^2$ and higher dimensions

The algorithm mentioned in the previous part gives constant factor approximation and achieves the linear run-time so the natural question is to analyze its extension to higher-dimensional datasets. In dimensions 2 and higher we cannot rely on the sorting the dataset points, so we need to consider several problems arising. In the Algorithm 1 on the line 11 we have defined the farthest point as either the leftmost or the rightmost point of the dataset. In the high-dimensional case, this doesn't hold anymore. But the Lemma 3 assures that we can keep constant approximation ratio, even if we choose initial center uniformly at random. Another solution is to use approximate furthest neighbor in high dimensions as defined in [5]. Another case where we used the fact that the input points were sorted in the one-dimensional case are lines 26-39 of Algorithm 1. Here, we traversed all the points that are less than βr far away from a given point by moving an index until we reach a squared distance bigger than the wanted radius. In high-dimensional cases, we can tackle this problem by using locality sensitive hashing as explained in [6]. Their algorithm achieves $O(n^\rho + d \log n)$ query time where $\rho \leq 7/(8c^2) + O(1/c^3) + o(1)$ and c is the approximation ratio.
The final problem to consider is how to calculate the value of a given ball (c, r). One of the approaches considered here is similar to one-dimensional case. We take a n-dimensional hypercube centered at c with side length $2r$ and calculate its value by using the same formula. For the two-dimensional case we obtain the following expression

$$value(A^*) = \sum_{z \in A^*} = \sum_{z \in A^*} r - d(z, x) =$$

$$\sum_{\substack{z \in \mathcal{X} \\ c_x - r \leq z_x \leq c_x + r \\ c_y - r \leq z_y \leq x_y + r}} r - ((c_x - z_x)^2 + (c_y - z_y)^2) =$$

$$|A^*| \cdot r - |A^*| \cdot c_x^2 - |A^*| \cdot c_y^2 - \Big(\sum_{\substack{z \in \mathcal{X} \\ c_x - r \leq z_x \leq c_x + r \\ c_y - r \leq z_y \leq x_y + r}} z_x^2 \Big) - \Big(\sum_{\substack{z \in \mathcal{X} \\ c_x - r \leq z_x \leq c_x + r \\ c_y - r \leq z_y \leq x_y + r}} z_y^2 \Big) +$$

$$(2c_x \sum_{\substack{z \in \mathcal{X} \\ c_x - r \leq z_x \leq c_x + r \\ c_y - r \leq z_y \leq x_y + r}} z_x) + (2c_y \sum_{\substack{z \in \mathcal{X} \\ c_x - r \leq z_x \leq c_x + r \\ c_y - r \leq z_y \leq x_y + r}} z_y)$$

where z_x is x coordinate of the point z and similar for the other points. Similarly as in the one-dimensional case, we will need to obtain the sum of coordinates of the points laying inside a n-dimensional hypercube. Thus, we reduced the problem to answering the range query in n-dimensional space. This is a well-studied problem and the algorithm with construction time

$O(n \log^d n)$ time and query time $O(\log^d n)$ is given by [7]. Overall running time of the preprocessing phase becomes $O(n\,d + n^{1+\rho} + n \log^d n)$ for constructing the LSH data structure and n-dimensional range tree. Further steps will require $O(n\,k\,d\,l\,(n^\rho + d \log n + \log^d n))$. Although the approximative estimation of ball value with a hypercube seems to work well in practice, we do not have any theoretical guarantees on how does it change the approximation ratio of the algorithm.

6 Conclusion

In Section 5.2 we described the possible improvements of the algorithm for *k-means* seeding. We have seen that the biggest issue arising is how to calculate the value of a ball. The analysis of the online median problem given in section 4.2 shows that we do not need to consider the exact values of balls in our algorithm. Instead, we only rely on the fact that our algorithm chooses the maximum value ball from a given set of balls. Thus, it would be interesting to further explore an efficient way to calculate (or estimate) ball values such that the relative ordering is (approximately) preserved. This would lead to an algorithm for *k-means* seeding with constant factor approximation ratio and time linear in n, k, and d.

References

[1] David Arthur, Sergei Vassilvitskii. *k-means++: the advantages of careful seeding.* SODA 2007: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, pages 1027-1035

[2] Ramgopal R. Mettu, C. Greg Plaxton. *The Online Median Problem.* SIAM Journal on Computing 2013, pages 816–832

[3] Sanjoy Dasgupta, Mohan Paturi. *Lecture notes for CSE 291: Geometric algorithms.* `https://cseweb.ucsd.edu/~dasgupta/291-geom/kmeans.pdf`

[4] Anup Bhattacharya, Regesh Jaiswal, Nir Ailon. *A tight lower bound instance for k-means++ in constant dimension.* TAMC 2014: Theory and Applications of Models of Computation. pages 7-22

[5] Rasmus Pagh, Francesco Silvestri, Johan Sivertsen, Matthew Skala. *Approximate Furthest Neighbor in High Dimensions.* SISAP 2015: Similarity Search and Applications. pages 3-14

[6] Alexandr Andoni, Piotr Indyk, Huy L. Nguyen, Ilya Razenshteyn. *Beyond locality-sensitive hashing.* SODA 2014: Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms, pages 1018-1028

[7] George S. Lueker. *A data structure for orthogonal range queries* SFCS 1978: Proceedings of the 19th Annual Symposium on Foundations of Computer Science, pages 28-34