Lecture Notes

Cluster Analysis

Dr. Melanie Schmidt

Department of Computer Science

University of Bonn

January 24, 2019

These lecture notes are written during the winter semester 2018/2019. Many thanks go to Carsten Fischer for proofreading the first draft of Chapter 2 and Clemens Rösner for thoroughly checking some of the proofs. Many thanks also go to all attendens of the class that helped to improve the lecture notes further by giving feedback and notifying me of typos (in class and by email), including Lena Carta, Jan Eube, Sirko Höer, Max Kanold, Joshua Könen, Manzil Roy and Julian Wargalla.

Please send further comments to `melanieschmidt@uni-bonn.de`.

Contents

Chapter 1

Introduction

[This chapter is under construction!]

Clustering is a fundamental unsupervised learning tool. It can be used to search data for hidden structure, to summarize data and to classify new unseen data according to prior findings. Since applications are manifold, clustering also offers a rich variety of algorithmic problems. In this lecture, we will study clustering problems through the lens of algorithm theory. Thus, we are equipped with a mathematical formulation that allows us to judge the quality of a clustering, and our goal is to find a clustering that has the best possible quality.

The clustering problems studied in this lecture share some properties: Most of them are *partitional* clustering problems, i.e., we get some input data and want to partition it into disjoint subsets of associated elements. Furthermore, most problems that we study are *center-based*, which means that we do not only want the partitioning, but we want a set of *centers* and an assignment of points to centers which then implicitly defines the partitioning. All clustering problems are *geometric* clustering problems, which means that we have geometric information about the input data (which we then call input *points*), i.e., we know the distances between any two points.

For our study of geometric clustering problems, we need to recall the definition of *metric spaces*. The most known metric space is the Euclidean space $\mathbb{R}^d$, where two points $x, y \in \mathbb{R}^d$ with $x = (x_1, \ldots, x_d)^T$ and $y = (y_1, \ldots, y_d)^T$ have distance $||x - y|| = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$. It is a special case of the following more general definition.

Definition 1. *Let X be any non-empty set and $d : X \times X \to \mathbb{R}$ be a function on X. Then d is* a metric on X *if it holds for all $x, y, z \in X$ that*

1. *$d(x, y) \geq 0$ and $d(x, y) = 0$ if and only if $x = y$,*
2. *$d(x, y) = d(y, x)$, and*
3. *$d(x, y) \leq d(x, z) + d(z, y)$.*

We call a pair (X, d) consisting of a finite set X and a metric d a metric space.

We refer to Condition 3 in Definition 1 as the *triangle inequality*. Usually we call elements of X *points*. Furthermore, we extend any metric d to point sets by setting

$$d(x, A) = \inf_{y \in A} d(x, y)$$

for any $A \subseteq X$ and

$$d(A, B) = \inf_{x \in A, y \in B} d(x, y).$$

for any $A, B \subseteq X$.

The Euclidean space $\mathbb{R}^d$ with the Euclidean distance is one example for a metric space. We will encounter it later in the lecture when we study the k-means problem. This modeling is common in machine learning and statistics, where k-means comes up as an important problem. However, clustering is also a classical topic in graph theory. Here, it is more common to study the case of *finite metric spaces*. The input then consists of a *finite* point set P and a distance function $d : P \times P \to \mathbb{R}$ which is metric, i.e. (P, d) form a metric space. In this scenario, d simply consists of $|P|^2$ numbers that give the distance for every pair of points in P. The can thus be viewed as an undirected complete weighted graph: the vertices are the points in P, and the weight of edge $\{x, y\}$ is $d(x, y) = d(y, x)$ for all $x, y \in P$.

In this context, the the metric space (P, d) can also be implicitly given by a *graph metric*: For any (not necessarily complete) graph $G = (V, E)$ weighted by a function $w : V \times V \to \mathbb{R}^{\geq 0}$, we define the *shortest path distance* $d_s(x, y)$ between $x, y \in V$ to be the length of the shortest path between x and y in G. One can show that d_s is a metric on V, and thus G and w implicitly define the metric space (V, d_s).

All problems studied in this lecture are minimization problems. As it turns out, for most of them, it is not known how to compute optimal solutions in polynomial time, and it may well be impossible to do so, since the problems are NP-hard. For this reason, we will resort to approximation algorithms that find a reasonably good solution instead of an optimal one.

Recall that an α-approximation algorithm for a minimization problem is an algorithm that outputs solutions with a value that is at most α times the value of an optimal solution. Let $\mathfrak{I}$ be the set of all possible input instances to an optimization problem, let ALG be an algorithm that computes a feasible solution $S^{ALG}(I)$ for any $I \in \mathfrak{I}$. For any $I \in \mathfrak{I}$, let $S^*(I)$ be an optimal solution for I. For a minimization problem, suppose that $c(S)$ is the cost of a solution S. Then ALG is an *α-approximation algorithm* if

$$\sup_{I \in \mathfrak{I}} \frac{c(S^{ALG}(I))}{c(S^*(I))} \leq \alpha.$$

Whenever we talk about α-approximation algorithms, we implicitly mean algorithms that have a polynomial worst-case running time. All problems considered in this lecture can be solved optimally given exponential running time.

Our studies will also include *randomized* approximation algorithms. For randomized algorithms, $c(S^{ALG}(I))$ or $v(S^{ALG}(I))$, respectively, is a random variable. We might either want to achieve that it is close to the optimum value with high probability, or

that its expected value is close to the optimum value. We choose the second alternative. Thus, we say that a randomized algorithm ALG is an *α-approximation algorithm in expectation* for a minimization problem if

$$\sup_{I \in \mathfrak{I}} \frac{\mathbf{E}\left[c(S^{ALG}(I))\right]}{c(S^*(I))} \leq \alpha.$$

As for deterministic algorithms, we also require randomized approximation algorithms to have a polynomial worst-case running time.

1.1 Mathematical facts and notations

This section collects mathematical facts that we need throughout the lecture.

Fact 2. *For all $r \in \mathbb{R}$ with $r \neq 1$ and all $n \geq 0$,*

$$\sum_{i=0}^{n} r^i = \frac{1 - r^{n+1}}{1 - r}.$$

If $|r| < 1$, then

$$\sum_{i=0}^{\infty} r^i = \frac{1}{1 - r}.$$

For a number $i \in \mathbb{N}^{\geq 1}$, we sometimes use the abbreviation $[i] = \{1, \ldots, i\}$ to denote the set of all numbers from 1 to i.

Euclidean geometry

Fact 3. *The Euclidean space $\mathbb{R}^d$ consists of all vectors $(x_1, \ldots, x_d)^T$ with $x_i \in \mathbb{R}$ for all $x_i \in \mathbb{R}$. The length of a vector $x \in \mathbb{R}^d$ is defined as*

$$||x|| := \sqrt{\sum_{i=1}^{d} x_i^2},$$

and this defines a norm. The distance between two points $x, y \in \mathbb{R}^d$ is the length of their difference vector, i.e.,

$$||x - y|| = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}.$$

This distance is a metric on $\mathbb{R}^d$.

A *matrix* $A \in \mathbb{R}^{n \times d}$ consists of $n \cdot d$ values from $\mathbb{R}$ that are organized into n rows and d columns. The Frobenius norm is an important norm for matrices.

Definition 4 (Frobenius norm)**.** *Let $A \in \mathbb{R}^{n\times d}$ be a matrix and let a_{ij} be the jth entry in row i of A. The* Frobenius norm *of A is denoted by $||A||_F$ and defined by*

$$||A||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{d} a_{ij}^2}\ .$$

For two matrices $A \in \mathbb{R}^{n\times d}$ and $B \in \mathbb{R}^{d\times m}$,

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & & a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{2n} & \cdots & a_{nd} \end{pmatrix} \text{ and } B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{d1} & b_{d2} & \cdots & b_{dm} \end{pmatrix},$$

the matrix product AB is defined to be the matrix $C \in \mathbb{R}^{n\times m}$ with

$$AB = C = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1m} \\ c_{21} & c_{22} & & c_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nm} \end{pmatrix} \text{ where } c_{ij} = \sum_{k=1}^{m} a_{ik}b_{kj}.$$

For two vectors $x, y \in \mathbb{R}^d$, the scalar product (or dot product) and the tensor product can be viewed as special cases of the matrix product for matrices of sizes $d \times 1$ and 1×1. Specifically, the *scalar product* of $x \in \mathbb{R}^d$ and $y \in \mathbb{R}^d$ is $\langle x, y\rangle := x^t y = \sum_{i=1}^{d} x_i y_i$, and the *tensor product* of $x \in \mathbb{R}^d$ and $y \in \mathbb{R}^d$ is

$$xy^T = \begin{pmatrix} u_1v_1 & \cdots & u_1v_d \\ \ldots & \ddots & \ldots \\ u_dv_1 & \cdots & u_dv_d \end{pmatrix}.$$

Notice that the result of the scalar product is a value in $\mathbb{R}$, and the result of the tensor product is a $(d \times d)$-matrix.

For a point $x \in \mathbb{R}^d$, the scalar product of x with itself gives the squared length of x, i.e., $\langle x, x\rangle = ||x||^2$. If the scalar product of two points $x, y \in \mathbb{R}^d$ is zero, then we say that they are *orthogonal*. We call a set of points *orthonormal* if all points are unit vectors (i.e., they have length 1), and they are pairwise orthogonal. Orthogonal (and in particular orthonormal) vectors satisfy the Pythagorean theorem.

Fact 5. *Let $x_1, \ldots, x_k \in \mathbb{R}^d$ be $k \leq d$ pairwise orthogonal vectors from $\mathbb{R}^d$. Then it holds that*

$$\left|\left|\sum_{i=1}^{k} x_i\right|\right|^2 = \sum_{i=1}^{k} ||x_i||^2.$$

A matrix is called *orthogonal* when the column vectors of the matrix are of unit length and pairwise orthogonal (they form a set of orthonormal vectors).

Chapter 2

The happy world of k-center

We start with the study of the *k-center problem*. It is a classical problem from graph theory, and it is one of the rare problems where we precisely know the best possible approximation ratio under the assumption that $P \neq NP$. This approximation ratio can be achieved by two elegant approximation algorithms.

2.1 The metric k-center problem

The input to the k-center problem is a finite metric space (P, d) and a number $k \in \mathbb{N}^{\geq 1}$. The k-center problem is a center-based clustering problem, meaning that we are searching for a set of centers C that give a good clustering when every point is assigned to the center closest to it. Center-based clustering problems differ in the evaluation of the quality of a center set. The k-center problem is a so-called min-max problem: We are minimizing the maximum distance of any point to its closest center.

Definition 6. *Let (P, d) be a metric space and $k \in \mathbb{N}^{\geq 1}$ be a number. The* k-center

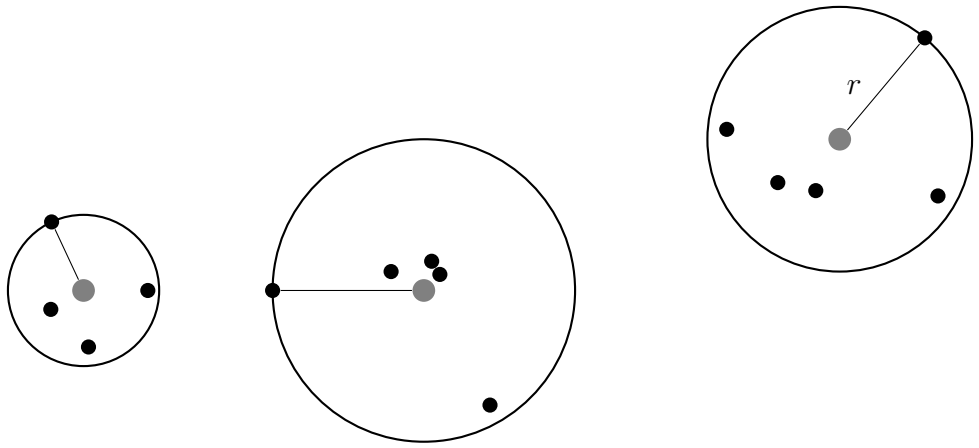


Figure 2.1: An example for a k-center instance and solution with $k = 3$.

problem *is to compute a set* $C \subseteq P$ *with* $|C| = k$ *that minimizes*

$$r(C) := \max_{x \in P} \min_{c \in C} d(x, c).$$

We call the value $\max_{x \in P} \min_{c \in C} d(x, c)$ either *k-center objective value* or *maximum radius* of the solution C. Figure 2.1 shows a point set, three chosen centers and the radii of the resulting three clusters. The maximum radius of the depicted solution is labeled by r.

2.2 A simple and elegant approximation

Gonzalez' algorithm [Gon85] gives a greedy 2-approximation for the k-center problem. The algorithm starts by picking an arbitrary point $c_1 \in P$ as the first center. Then it does $k-1$ iterations of the following: Choose the point c_i which is farthest away from the previously chosen centers $\{c_1, \ldots, c_{i-1}\}$ to be the center. This is repeated until k centers are chosen.

`farthest-first-traversal(`a finite metric space (P, d)`,` $k \in \mathbb{N}^{\geq 1}$`)`

1. Choose a center c_1 arbitrarily from P and set $C^1 = \{c_1\}$
2. **For** $i = 2$ **to** k **do**
3. Choose c_i as (one of) the points maximizing $d(x, C^{i-1})$, i.e., $c_i = \arg\max\{d(x, C^{i-1}) \mid x \in P\}$
4. Set $C^i = C^{i-1} \cup \{c_i\}$
5. **Return** $C = \{c_1, \ldots, c_k\}$

Figure 2.2 shows an example run of Algorithm `farthest-first-traversal` with $k = 3$. The first center is chosen arbitrarily. Center 2 and 3 are then chosen as points with maximum distance to the previously chosen centers. This means that the algorithm has the tendency to choose points that are at the border of a cluster. The resulting solution is worse than the solution depicted in Figure 2.1, but its maximum radius r' only slightly larger than r from the previous solution. The following theorem shows that in general, the maximum radius is at most twice the maximum radius of an optimal solution.

Theorem 7. *Algorithm* `farthest-first-traversal(`$(P, d), k$`)` *computes a* 2*-approximation for the k-center problem.*

Proof. Let C be the solution computed by `farthest-first-traversal(`$(P, d), k$`)`. This solution has radius $r(C) = \max_{x \in P} d(x, C)$. Then there is a point $y \in P$ that actually has distance $r(C)$ to its closest center in C, i.e., $d(y, C) = \min_{c \in C^k} d(y, c) = r(C)$. Now we observe that the distance between y and its closest center can only get shorter during the execution of the algorithm, i.e., $d(y, C^i) \geq d(y, C^k)$ for all $i \in \{1, \ldots, k\}$. Thus, $d(y, C^i) \geq r(C)$ is true for all $i \in \{1, \ldots, k\}$. Since Gonzalez'

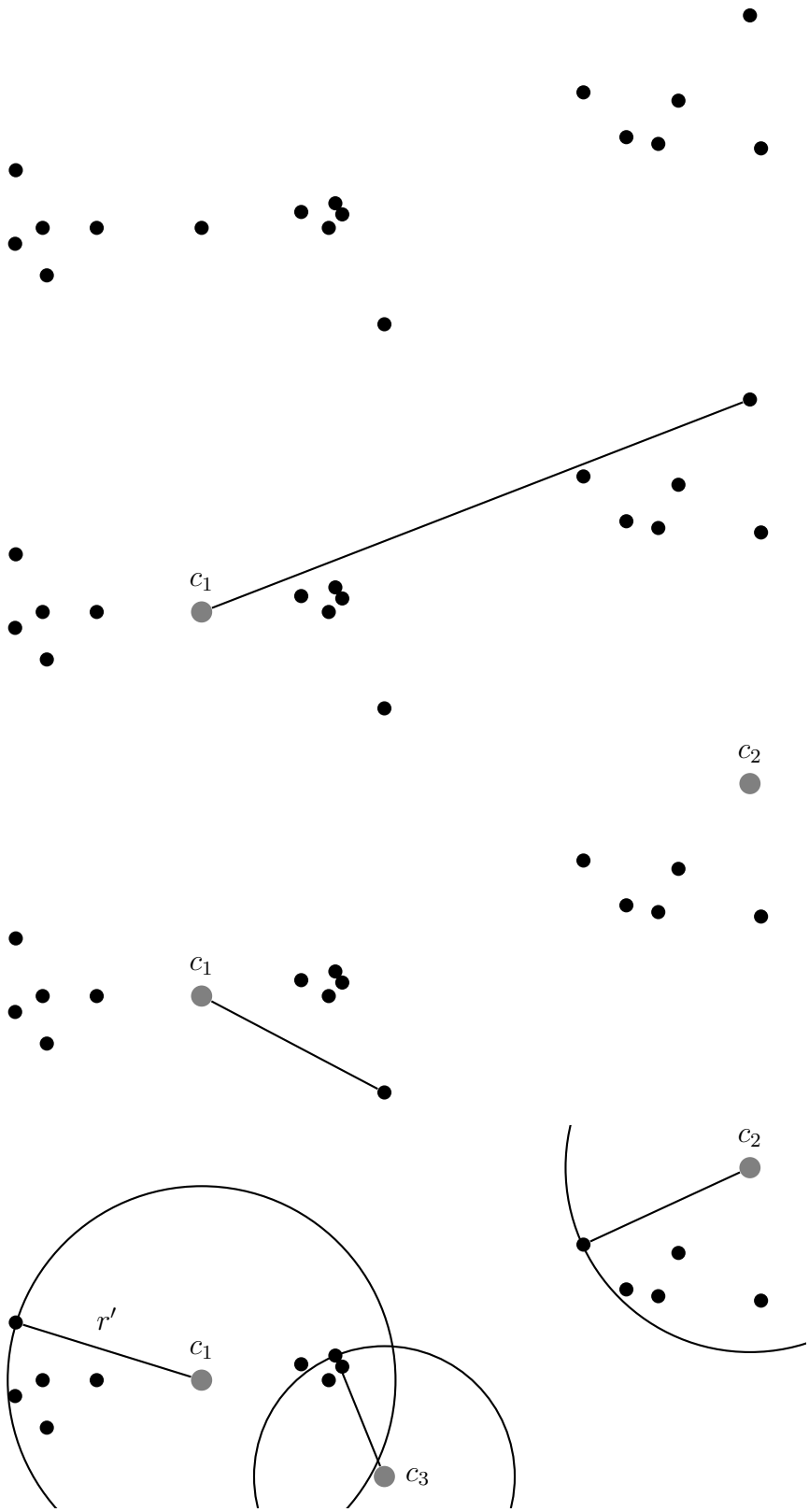


Figure 2.2: An example run of farthest first traversal with $k = 3$.

algorithm always chooses the point that has maximum distance to the previously chosen centers, $d(y, C^i) \geq r(C)$ implies $d(c_{i+1}, C^i) \geq r(C)$ for all $i \in \{1, \ldots, k-1\}$. This means that the centers and y form a set of $k+1$ points with pairwise distance $r(C)$. Any solution has to have a cluster with two of these points, and this cluster then has at least radius $r(C)/2$. Since the optimal solution has a maximum radius of at least $r(C)/2$, and the computed solution has a radius of $r(C)$, we conclude that the computed solution is a 2-approximation. □

Lemma 8. *Algorithm* `farthest-first-traversal`(P, d) *can be implemented to run in time* $\mathcal{O}(|P| \cdot k)$.

Proof. Line 1 takes constant time. The loop in lines 2-4 is executed $k-1$ times. For line 3, the algorithm needs to find the point whose closest center is farthest away. To keep track of this, we use an array $c_{\min}$ which stores the closest center (so far) for each point, i.e. we preserve the equality $c_{\min}(x) = \arg\min\{c \in C^{i-1} d(x, c)\}$ for all $x \in P$. The array is initialized in time $\mathcal{O}(|P|)$ before starting the loop. In the loop, when we choose a new center c_i, we compare $d(x, c_i)$ with $d(x, c_{\min}(x))$ and update $c_{\min}(x)$ accordingly. This takes constant time for every point, i.e., updating $c_{\min}$ takes time $\mathcal{O}(|P|)$. We can find the point with the maximum $c_{\min}$ value simultaneously or find it in another $\mathcal{O}(|P|)$ operations after the updating is finished. Thus, lines 2-4 take time $\mathcal{O}(|P|\dot{k})$. □

Was part of lecture 1, but is not used until later

The following definitions will later be part of the respective chapters.

Definition 9. *Let* (P, d) *be a metric space and* $f : P \to \mathbb{N}^{\geq 1}$ *be a function of* opening costs. *The* facility location problem *is to compute a set* $C \subseteq P$ *that minimizes*

$$\sum_{x \in P} \min_{c \in C} d(x, c) + \sum_{c \in C} f(c).$$

– end of lecture 1 –

2.3 A matching lower bound

The k-center problem is NP-hard. This has been observed simultaneously by Hochbaum [Hoc84] and Hsu and Nemhauser [HN79], who also both discovered the inapproximability result that we will see below. Both their proofs rely on a reduction from the *dominating set problem*.

Definition 10. *Let* $G = (V, E)$ *be an undirected unweighted graph. A set* $D \subset V$ *is a* dominating set *for* G *if every vertex in* V *has a neighbor in* D*, i.e., for all* $u \in V$*, either* $u \in D$ *or there exists a* $v \in D$ *such that* $\{u, v\} \in E$.

The dominating set problem *is to decide for a given graph* G *and a number* $k \in \mathbb{N}^{\geq 1}$ *whether a dominating set* D *for* G *exists that has cardinality at most* k*, i.e.,* $|D| \leq k$.

The dominating set problem is NP-hard; it is one of the problems shown to be NP-hard in the book by Garey and Johnson [GJ79]. It is closely related to the k-center problem.

Lemma 11. *The k-center problem is NP-hard. Furthermore, it is NP-hard to compute an α-approximation for the k-center problem for any $\alpha < 2$.*

Proof. Let $G = (V, E)$, k be an input to the dominating set problem. We define an input for the k-center problem by setting $P = V$ and setting the distance $d(x, y)$ between $x \in P$ and $y \in P$ to 1 if $\{x, y\} \in E$, and to 2 otherwise. This is a metric because no function $d : P \times P \to \{1, 2\}$ can violate the triangle inequality. Assume that we are given an algorithm that decides the decision variant of the k-center problem, i.e., for a given weighted graph, a number of centers and a value t, it decides whether a solution to the k-center problem of radius at most t exists. We apply this algorithm to the input P, d, k and $t = 1$. We output `YES` if the algorithm finds a clustering of radius at most 1 and `NO` otherwise.

Now we argue that this reduction is correct. First, assume that a dominating set D of size at most k exists in D. Then every vertex in V is either in D or it has a neighbor in D. Consider D as a center set in G'. Then every point in P either has distance 0 to its closest center (if it is in D itself), or it has distance 1 (if it has a neighbor in D). Thus, the maximum radius of the solution is 1. Second, assume that the algorithm found a center set C with maximum radius 1 in G'. Then C is a dominating set in G: Every vertex has distance ≤ 1 to at least one vertex in C, so it is either in C or a neighbor of a point in C.

This shows that the k-center problem is NP-hard. But that is not all: We observe that the reduction nearly immediatly also gives an inapproximability result. That is because the only possible distances between two distinct points in G' are 1 or 2. Thus, any approximation algorithm with an approximation ratio $\alpha < 2$ has no other option than providing an optimal solution. More precisely we can formulate the above reduction in the following way: Assume that we are given an algorithm that computes an α-approximation for the k-center problem for $\alpha < 2$. Apply this algorithm to P, d, k. If the maximum radius of the solution is 1, then output `YES`, otherwise, output `NO`. By the same arguments as above, this reduction is correct and shows that it is NP-hard to compute an α-approximation for the k-center problem for $\alpha < 2$. □

2.4 Incremental and hierarchical clustering

In this section, we consider *incremental* and *hierarchical* clustering. The basic idea behind these notions is that instead of one clustering for one fixed number of centers, we want multiple clusterings, indeed, even *one clustering for each possible value of k*. The two notions are related, but differ in the details of the desired clusterings.

Incremental clusterings are motivated by the scenario where we want to place centers, but we do not know how many we will build, or we cannot build them all at the same

time. An incremental clustering thus gives an ordering of the input points, and the first k points in this order form the center set for the k-center clustering with k centers.

In the following, we will compare incremental and hierarchical clusterings with optimal clusterings for a fixed number of centers. For this purpose we refer to an optimal solution for the k-center problem (with k centers) by C_k^{opt}.

Definition 12. *Let (P, d) be a metric space. An* incremental clustering *is an ordering $x_1, \ldots, x_{|P|}$ of all points in P.*

For any $k \in [|P|]$, the solution to the k-center problem on P induced by such an incremental clustering is the center set $\{x_1, \ldots, x_k\}$.

We say that an incremental clustering is α-competitive *if for all $k \in [|P|]$,*

$$r(\{x_1, \ldots, x_k\}) \leq \alpha \cdot r(C_k^{\text{opt}}),$$

where C_k^{opt} is an arbitrary optimum solution with k centers.

Our notion of competitiveness in Definition 12 compares the individual clusterings with optimal clusterings with the same number of clusters. Notice that we do not actually know the values $r(C_k^{\text{opt}})$, and can in general not compute them (assuming $P \neq NP$), so we can not compute the competitiveness of a given incremental solution.

In general, a 1-competitive incremental clustering does not even exist since the optimal clusterings may contradict each other in their choice of centers.

However, we observe that Gonzalez' algorithm can be used to compute an incremental clustering if we simply continue using it until all points have become a center. This means that the approximation algorithm which is optimal under the condition $P \neq NP$ can directly be used to compute incremental clusterings of the same quality.

`full-farthest-first-traversal(`a finite metric space (P, d)**`)`**

1. Choose a center c_1 arbitrarily from P, set $C^1 = \{c_1\}$
2. **For** $i = 2$ **to** $|P|$ **do**
3. Set $R_i := \max_{x \in P} d(x, C^{i-1})$
4. Choose a c_i as a point with $d(c_i, C^{i-1}) = R_i$, set $C^i = C^{i-1} \cup \{c_i\}$
5. **Return** $c_1, \ldots, c_{|P|}$ and $R_2, \ldots, R_{|P|}$

The only real change compared to `farthest-first-traversal` is that the algorithm `full-farthest-first-traversal` does not have k as a parameter, runs the main loop until $|P|$ centers are chosen and returns the numbering instead of one fixed center set. The other changes are for our convenience: We name the distance of center c_i to the previously chosen centers R_i. This will prove helpful later.

Corollary 13. *The algorithm `full-farthest-first-traversal(`(P, d)`)` computes a 2-competitive incremental k-center clustering. In particular, $R_k \leq 2 \cdot r(C_{k-1}^{\text{opt}})$ holds for all $k \in \{2, \ldots, |P|\}$ and optimal solutions C_k^{opt}.*

Proof. This follows directly from Theorem 7. □

Later in this lecture, we see that it is a lucky coincidence that a best-possible approximation algorithm for the k-center problem already computes an incremental solution. For other objectives, this is not the case, and obtaining incremental clusterings is much more involved.

Hierarchical clusterings are related to incremental clusterings, yet they are motivated differently. Here, the *clusterings* have to be related. More precisely, we want that for all k, the clustering with k clusters can be obtained from the clustering with $k+1$ clusters by merging two of the clusters. We call this *hierarchical compatibility*. Until now, we have had a fairly center-based view on clusterings. That makes sense in all scenarios where we want to summarize the data by the centers, and thus, the centers are really what we are interested in. However, in some scenarios it is the *partitioning itself* that we are curious about. We want to know how the data points themselves cluster into groups, and the center is just a means to judge the (dis)similarity of the points in the same cluster. This is the case for hierarchical clustering. We want to see the data clustered into groups at different levels of granularity.

Definition 14. *Let (P, d) be a metric space. A* hierarchical k-center clustering *consists of $|P|$ partitionings $\mathcal{H}_1, \dots, \mathcal{H}_{|P|}$ of P that satisfy*

- *$|\mathcal{H}_k| = k$ for all $k \in \{1, \dots, |P|\}$, in particular this means that $\mathcal{H}_1 = \{P\}$ and $\mathcal{H}_{|P|} = \{\{x\} \mid x \in P\}$,*
- *and $\mathcal{H}_k = \mathcal{H}_{k-1} \backslash \{A\} \cup \{A_1, A_2\}$ for all $k \in \{2, \dots, |P|\}$, with $A \in \mathcal{H}_{k-1}$ and two clusters $A_1, A_2 \in \mathcal{H}_k$ with $A = A_1 \dot\cup A_2$.*

Let $\mathcal{H}_k$ be one of the clusters in the hierarchy, and assume that we name its clusters $C_1, \dots, C_k$. Then we define $r(\mathcal{H}_k)$ to be the maximum radius of the clusters if we choose centers optimally, i.e., $r(\mathcal{H}_k) := \max_{i \in [k]} \min_{c \in C_i} \max_{x \in C_i} d(x, c)$.

We say that an hierarchical clustering $\mathcal{H}_1, \dots, \mathcal{H}_{|P|}$ is α-competitive *if*

$$r(\mathcal{H}_k) \leq \alpha \cdot r(C_k^{\text{opt}})$$

for all $k \in [|P|]$ and optimal solutions C_k^{opt}.

Definition 14 is fairly complex. Each $\mathcal{H}_k$ is a *partitioning* of P, i.e., it contains k sets that are disjoint and whose union is P. To compute the cost of such a clustering, we consider optimal centers: For each cluster in a clustering, we pick the point in the cluster which minimizes the 1-center cost of that cluster. These centers may change between the levels of the hierarchy. So our notion of hierarchical clustering is different from incremental clustering in two ways: Firstly, we require the clusters themselves to be hierarchically compatible, which is a stronger requirement than having nested center sets. But secondly, we do not require the centers to stay the same, so here we require less. For each level, we may completely change our choice of centers.

Gonzalez' algorithm will again be useful for our goal to find a good hierarchical clustering. However, this time the farthest first traversal is only the first building block of

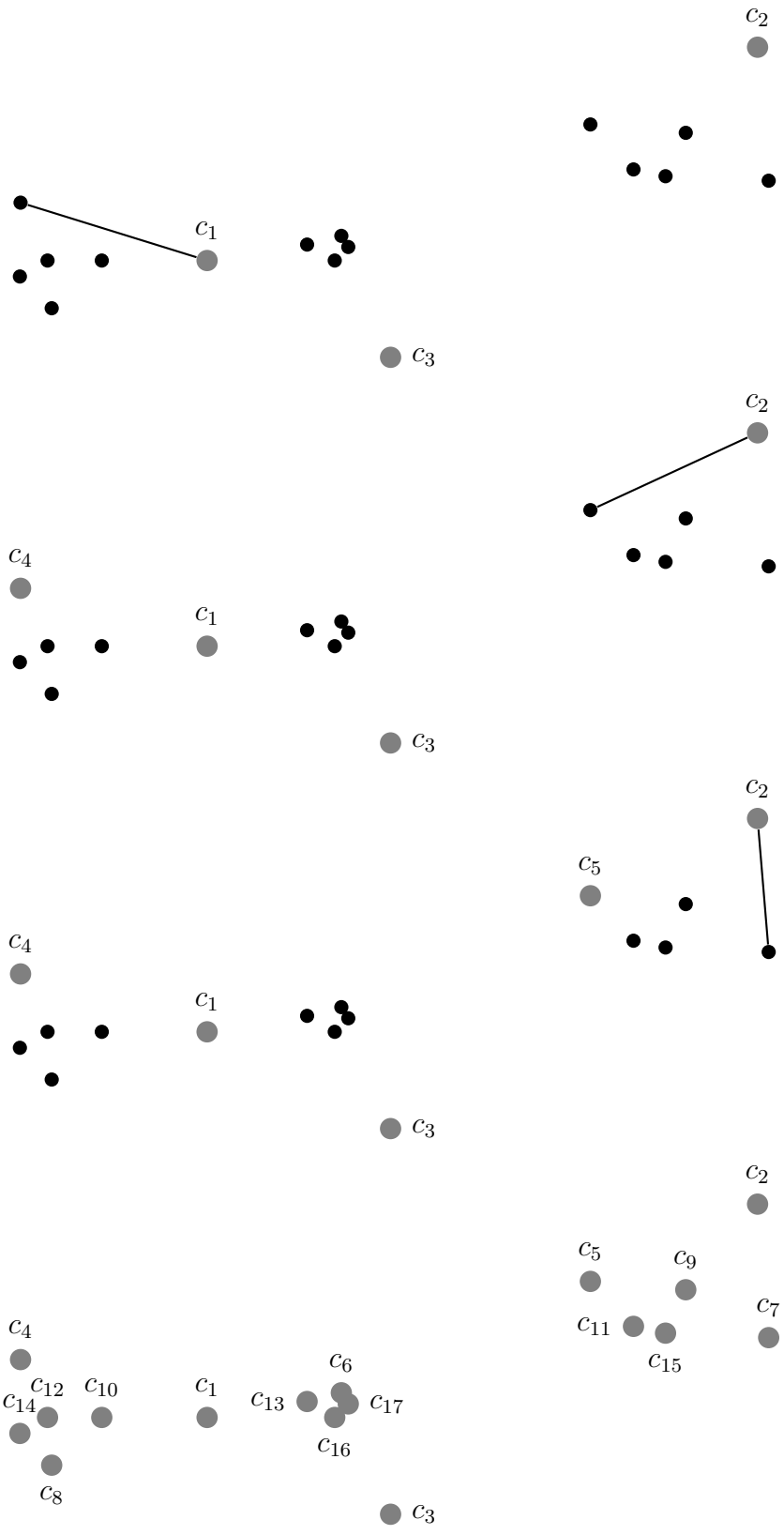


Figure 2.3: Continuation of Figure 2.2 as an example for a full farthest first traversal.

the algorithm. The incremental clustering computed by Gonzalez' algorithm has the properties that we investigated above, in particular, $\{x_1, \ldots, x_j\}$ is a 2-approximative solution for the k-center problem with $k = j$. However, the clusters induced by $\{x_1, \ldots, x_j\}$ for different values of j are in general not hierarchically compatible with each other. For example, the cluster that was associated with x_j in the j-clustering may split up when assigned optimally to the first $j - 1$ centers.

Still, we will make use of the incremental clustering. Say that the full farthest first traversal outputs the order $x_1, \ldots, x_{|P|}$ and the values $R_2, \ldots, R_{|P|}$. We now find a good hierarchical clustering in three more steps.

I. Levels of granularity The full farthest first traversal provides us with $|P|$ clusterings of decreasing cost: The first one is the 1-center clustering where every point is assigned to x_1, the second clustering is induced by assigning every point to either x_1 or x_2, and so on. The costs of these solutions are $R_2, \ldots, R_{|P|}$ for $k = 1, \ldots, |P| - 1$. [Recall that R_i is the cost of the $(i-1)$-clustering, since it is the maximum distance between any point to the centers in $\{x_1, \ldots, x_{i-1}\}$.] The clustering for $k = |P|$ has cost 0 since every point has its own cluster. We now group the points into levels with the goal that the corresponding R_i values have roughly the same cost for points in the same level. The largest R_i value is R_2, and we also call this value R, i.e., $R = R_2$. The formal definition of the levels is:

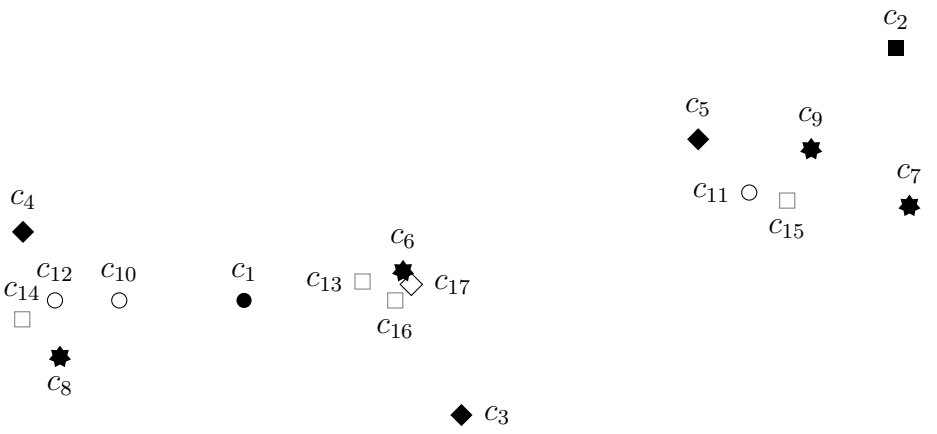
$$L_0 = \{x_1\}, \quad L_j = \left\{ x_i \mid R_i \in \left(\frac{R}{2^j}, \frac{R}{2^{j-1}} \right] \right\} \text{ for all } j \geq 1.$$

The first point x_1 is special because it does not have an R_i value (these start at $i = 2$). Thus, x_1 gets its own special level L_0. The remaining points are partitioned into the levels according to their R_i in an exponentially decreasing manner, based on $R = R_2$. L_1 contains all points x_i with $R/2 < R_i \leq R$. This in particular includes x_2 since $R_2 = R$, and it may or may not include further points. L_2 contains all x_i with $R/4 < R_i \leq R/2$, and in general, L_j contains all x_i with $R/2^j < R_i \leq R/2^{j-1}$. Figure 2.4 depicts the resulting levels for the points in our ongoing example.

Notice that we do not give an upper bound on j, i.e., on the maximum level index. Indeed, we do not know any such bound: The cost of the $(|P| - 1)$-center clustering may be arbitrarily small compared to the cost of the 1-center clustering. However, at most $|P|$ of the levels are non-empty because there are only $|P|$ points that are assigned to levels. Furthermore notice that for all points in the same level, the R_i value is of a comparible magnitude, more precisely, they are all within a factor of 2 of the left interval boundary.

For use in the next paragraph, we define a function $L : \{x_1, \ldots, x_{|P|}\} \to \mathbb{N}^{\geq 1}$ which gives the index of the level that a point is in, i.e., $L(x_i) = j$ when $x_i \in L_j$.

II. The parent function We need the levels of granularity in order to define a mapping that we call the *parent function*. It assigns a parent to every point in P. For



● $L_0 = \{c_1\}$ ■ $L_1 = \{c_2\}$ ◆ $L_2 = \{c_3, c_4, c_5\}$ ✸ $L_3 = \{c_6, c_7, c_8, c_9\}$

○ $L_4 = \{c_{10}, c_{11}, c_{12}\}$ □ $L_5 = \{c_{13}, c_{14}, c_{15}, c_{16}\}$ ◇ $L_6 = \{c_{17}\}$

Figure 2.4: Continuation of Figure 2.3, showing the levels of all points.

every $i \in \{2, \ldots, |P|\}$, we define

$$\text{parent}(x_i) := \arg\min\{d(x_i, y) \mid y \in \bigcup_{j=0}^{L(x)-1} L_j\}.$$

For x_1, we set $\text{parent}(x_1) := x_1$, so x_1 is mapped to itself. For all $x_i \neq x_1$, the parent of x_i is its closest point among all points in a level with lower index than $L(x_i)$. Notice that this means that the parent is a point x_ℓ with $\ell < i$, and that among all points x_ℓ with $\ell < i$ we exclude exactly those that are in the same level as x_i itself. In our example, the parents are as follows:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$\text{parent}(x_i)$	–	1	1	1	2	3	2	4	5	8	5	8	6	12	11	6	6

Notice that the parent does not have to be in the level directly before the level of x, it can be in any level with a lower index. For example, the parent of $c_{16} \in L_5$ in our example is $c_6 \in L_3$. This is important, since it allows us to show the following bound for the distance between a point and its parent.

Lemma 15. *For all $x \in P$,*

$$d(x, \bigcup_{j'=0}^{j} L_{j'}) \leq \frac{R}{2^j} \quad \textit{and} \quad d(x, \text{parent}(x)) \leq \frac{R}{2^{L(x)-1}}.$$

Proof. This is a property of the farthest first traversal. Let x_ℓ be the point with highest index in L_j. Then all points are at distance $R_{\ell+1}$ from $\{x_1, \ldots, x_\ell\} = \bigcup_{j'=0}^{j} L_{j'}$, because

$R_{\ell+1}$ is the distance that $x_{\ell+1}$ had to this solution. Furthermore, $x_{\ell+1}$ is not in L_j, so its distance $R_{\ell+1}$ is at most $R/(2^j)$. We conclude that

$$d(x, \bigcup_{j'=0}^{j} L_{j'}) \leq R_{\ell+1} \leq \frac{R}{2^j}$$

for all $x \in P$. Furthermore, parent(x) is the closest point to x in $\bigcup_{j=0}^{L(x)-1} L_j$, so it follows directly that

$$d(x, \text{parent}(x)) = d(x, \bigcup_{j'=0}^{L(x)-1} L_{j'}) \leq \frac{R}{2^{L(x)-1}}$$

is true for all $x \in P$. □

III. The full algorithm The final algorithm computes the above described levels and the parent function, and then uses the parent function to compute a hierarchical clustering. This is the full pseudo code:

`hierarchical-kcenter(`a finite metric space (P, d)**`)`**

1. Compute $x_1, \ldots, x_{|P|}$ and $R_2, \ldots, R_{|P|}$ by calling `full-farthest-first-traversal`$((P, d))$
2. Set $R := R_2$ and set $L_0 := \{x_1\}$, $L(x_1) = 0$
3. **For all** $j \geq 1$, set $L_j = \{x_i \mid R_i \in (\frac{R}{2^j}, \frac{R}{2^{j-1}}])\}$, and set $L(x_i) = j$ iff $x_i \in L_j$
4. **For all** $x \in P$, set parent$(x) := \arg\min\{d(x, y) \mid y \in \bigcup_{j=0}^{L(x)-1} L_j\}$
5. Set $C_i = \{x_i\}$ and $\mathcal{H}_{|P|} = \{C_i \mid i \in [|P|]\}$
6. **For** $k = |P| - 1$ to 1 **do**
7. Notice that x_{k+1} is the center of the cluster C_{k+1} to be reassigned
8. Let $x_p = \text{parent}(x_{k+1})$ be the parent of x_{k+1}, notice $x_p \in C_p$
9. Set $C_p = C_p \cup C_k$ and $\mathcal{H}_k = \{C_i \mid i \in [k]\}$
10. **Return** $\mathcal{H}_1, \ldots, \mathcal{H}_{|P|}$

In Lines 1-4, the algorithm computes the incremental clustering by using Gonzalez' algorithm, then derives the levels of all points and computes the parent function. The remaining steps compute the actual hierarchical clustering. The process starts with $\mathcal{H}_{|P|}$, the partitioning into singleton clusters, i.e., into $C_i = \{x_i\}$. Now the algorithm creates the clusterings with less clusters iteratively. It always removes the last clustering (according to the numbering), and merges it with a clustering that is lexicographically before it. This is done by considering the parent $x_p = \text{parent}(x_{j+1})$ and adding the points in C_j to C_p (which is the cluster that contains x_p and was originally initialized by $C_p = \{x_p\}$). The cluster C_{j+1} is then dissolved, and the clustering with i clusters consists of the updated clusters $C_1, \ldots, C_j$. Figure 2.5 shows the hierarchical clustering that this procedure creates for our example point set (notice

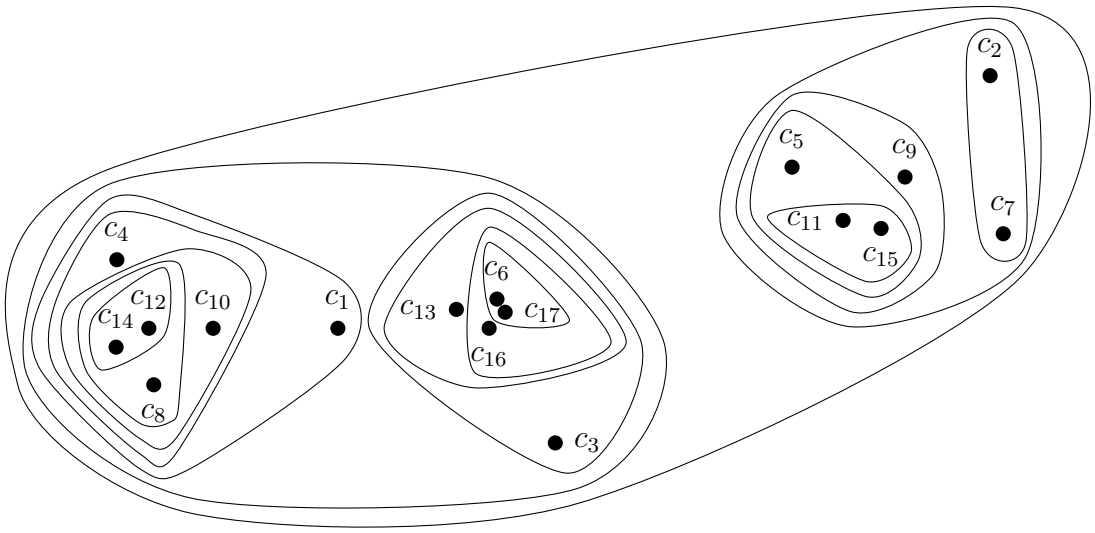


Figure 2.5: Continuation of Figure 2.4, depicting the resulting hierarchical clustering.

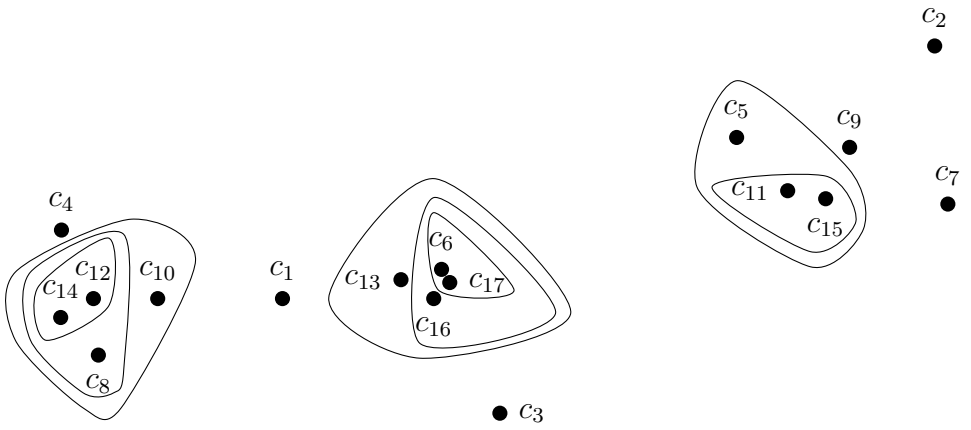


Figure 2.6: Continuation of Figure 2.5, depicting the clustering for $k = 9$.

that the picture does not uniquely define all intermediate clusterings, although the algorithm does, since the order is not depicted for unrelated sets). Figure 2.6 shows one specific clustering from the hierarchy, namely the clustering for $k = 9$.

– end of Lecture 2 –

By Lemma 15, we know that the first time that a point is reassigned (then, to its parent), the distance to its new center is relatively small: It is at most $R/(2^{L(x)-1})$. Before we proceed, let us briefly discuss what this means. Assume that $x = x_i$. Since x is on level $L(x)$, we know that $R_i \in (\frac{R}{2^{L(x)}}, \frac{R}{2^{L(x)-1}}]$. Now notice that R_i is the cost of the Gonzalez' solution for $k = i - 1$. At the point when we reassign $x = x_i$ for the first time, we compute the $(i-1)$-clustering. The cost of this clustering is R_i. We conclude that the $(i-1)$-clustering of Gonzalez costs $R_i > \frac{R}{2^{L(x)}} = \frac{1}{2} \cdot (R/(2^{L(x)-1})) \Leftrightarrow (R/(2^{L(x)-1})) \leq 2R_i$. So, compared to the Gonzalez' solution, x now has at most twice the distance to its closest center in the $(i-1)$-clustering. [Since Gonzalez' itself is a

2-approximation, this means that the distance between x and its closest center is at most 4 times the optimum maximum radius.]

Now the crucial question is: What does it mean for the radius that points are potentially reassigned multiple times? The key feature of the algorithm is that two solutions in different levels differ exponentially with respect to their cost. Thus, if a point is reassigned again and again, then the additional distance incurred by the first reassignments plays a smaller and smaller role compared to the later clusterings. Consider c_{15} in our ongoing example. Its parent is c_{11}, the parent of c_{11} is c_5, the parent of c_5 is c_2, and the parent of c_1 is c_1. This means that c_{15} is reassigned four times. Say we are interested in the clustering for $k = 4$, which is created when c_5 stops being a center, which in particular means that c_{15} is reassigned for the third time, namely to c_2. By the triangle inequality, we know that

$$d(c_{15}, c_2) \leq d(c_{15}, c_{11}) + d(c_{11}, c_5) + d(c_5, c_2).$$

Furthermore, we know that $c_{15} \in L_5$, $c_{11} \in L_4$, $c_5 \in L_2$ and $c_2 \in L_1$, implying that

$$\begin{aligned} & d(c_{15}, c_{11}) + d(c_{11}, c_5) + d(c_5, c_2) \\ \leq & \frac{R}{2^{L(c_{15})-1}} + \frac{R}{2^{L(c_{11})-1}} + \frac{R}{2^{L(c_5)-1}} \\ \leq & \frac{R}{2^4} + \frac{R}{2^3} + \frac{R}{2^1} \leq R. \end{aligned}$$

Since $c_5 \in L_2$, $R_5 \in (\frac{R}{2^2}, \frac{R}{2^1}]$, meaning that the 4-clustering costs at least $R/4$. So even though we did three reassignments, the bound only doubled compared to the distance between c_5 and its direct parent. The following theorem shows that in general, the solution for any k has a radius which is at most 4 times the radius of Gonzalez' solution with the same number of clusters. Since Gonzalez' is guaranteed to be a 2-approximation, we conclude that the hierarchical k-center algorithm is 8-approximative.

Theorem 16. *Algorithm* `hierarchical-kcenter`*$((P, d))$ computes a hierarchical clustering on (P, d) which is 8-approximative as defined in Definition 14.*

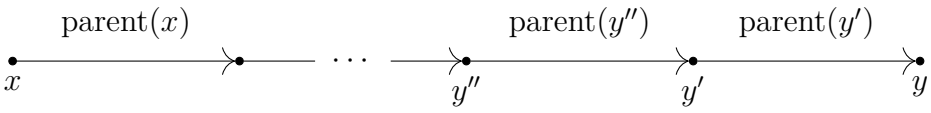
Proof. The output of Algorithm `hierarchical-kceter` is a hierarchical clustering by definition. We now fix a number $k \in \mathbb{N}^{\geq 1}$ and show that the k-clustering $\mathcal{H}_k$ contained in the hierarchy has a maximum radius which is at most 4 times as large as the maximum radius of the k-clustering computed by Gonzalez' algorithm.

To show this, we briefly use an extension of the parent function which we define inductively. The function parent^1 is just the parent function, defined by $\text{parent}^1(x) = \text{parent}(x)$ for all $x \in P \backslash \{x_1\}$, and $\text{parent}(x_1) = x_1$. The function parent^0 is the identity, i.e., $\text{parent}^0(x) = x$ for all $x \in P$. The function parent^i for $i > 1$ results from applying the parent function i times. More precisely, parent^i results from parent^{i-1} by setting $\text{parent}^i(x) = \text{parent}(\text{parent}^{i-1}(x))$ for all $x \in P$ with $x \neq x_1$ and $\text{parent}^i = x_1$.

Claim 17. *Let $x \in P$, $i \geq 1$, $y' = \text{parent}^{i-1}(x)$ and $y = \text{parent}(y') = \text{parent}^i(x)$. Then*

$$d(x, y) \leq \frac{R}{2^{L(x)-1}} + \frac{R}{2^{L(x)-2}} + \frac{R}{2^{L(x)-3}} + \ldots + \frac{R}{2^{L(y')-1}} = \sum_{j=L(y')}^{L(x)} \frac{R}{2^{j-1}}.$$

Proof. We prove the claim inductively. For $i = 1$, the claim follows from Lemma 15. Assume that that $i \geq 2$ and that the claim is true for $i-1$. If $\text{parent}^{i-1}(x) = \text{parent}^{i}(x)$, then we are done by induction hypothesis. Otherwise, set set $y'' = \text{parent}^{i-2}(x)$, thus we also know that $\text{parent}(y'') = y'$. Notice that y is in a lower level than y', i.e. $L(y) < L(y')$, by the definition of the parent function, and similarly, $L(y') < L(y'')$.



By induction hypothesis,

$$d(x, y') \leq \frac{R}{2^{L(x)-1}} + \frac{R}{2^{L(x)-2}} + \frac{R}{2^{L(x)-3}} + \ldots + \frac{R}{2^{L(y'')-1}}.$$

By the triangly inequality, $d(x, y) \leq d(x, y') + d(y', y)$. By Lemma 15, $d(y', y) \leq \frac{R}{2^{L(y')-1}}$. Combining these statements shows the claim. □

We want to bound the maximum radius of the k-clustering. This clustering results from reassigning the points $x_{|P|}, \ldots, x_{k+1}$, potentially multiple times. Let x be one of these points, and let y' be the last point among its parents which is in $\{x_{|P|}, \ldots, x_{k+1}\}$, and $y = \text{parent}(y')$ be the first one that is in $\{x_k, \ldots, x_1\}$. By Claim 17, we know that $d(x, y) \leq \sum_{j=L(y')}^{L(x)} \frac{R}{2^{j-1}}$. We can simplify this by using Fact 2:

$$d(x, y) \leq \sum_{j=L(y')}^{L(x)} \frac{R}{2^{j-1}} \leq \frac{R}{2^{L(y')-1}} \sum_{j=0}^{\infty} \left(\frac{1}{2}\right)^j = \frac{R}{2^{L(y')-1}} \cdot \frac{1}{1/2} = \frac{R}{2^{L(y')-2}}.$$

The radius of the k-clustering $\{x_1, \ldots, x_k\}$ is R_{k+1}. Now the crucial observation is that since y' is still in $\{x_{|P|}, \ldots, x_{k+1}\}$, its level $L(y')$ is at least $L(x_{k+1})$, i.e., $L(y') \geq L(x_{k+1})$, and

$$d(x, y) \leq \frac{R}{2^{L(y')-2}} \leq \frac{R}{2^{L(x_{k+1})-2}}.$$

Since $R_{k+1} \in (\frac{R}{2^{L(x_{k+1})}}, \frac{R}{2^{L(x_{k+1})-1}}]$, we get that

$$d(x, y) \leq \frac{R}{2^{L(x_{k+1})-2}} = 4 \cdot \frac{R}{2^{L(x_{k+1})}} < 4 \cdot R_{k+1}.$$

This is true for all points $x \in P$, thus the maximum radius of the solution is bounded by 4 times the maximum radius of the Gonzalez' solution, which is at most 2 times the maximum radius of an optimal solution. The theorem follows. □

2.5 Another elegant 2-approximation

In this section, we see another 2-approximation algorithm for the k-center problem. It is a little more complicated to state, but will give us new insights to the problem. It is based on the following crucial observation.

Observation 18. *There are only $\mathcal{O}(|P|^2)$ possible values for the maximum radius of the optimal solution for the k-center problem on (P,k).*

For any set P and any set of centers $C \subset P$, the k-center objective value is a *distance* between a pair of points, and there are only $\mathcal{O}(|P|^2)$ such pairs. Thus, we can 'guess' the optimum value. This is a technique often applied: One shows that the number of possible values for some quantity is bounded, iterates through all of them and can then perform an algorithm with the assumption that the current guess is correct.

In the case of k-center we use our guess to build a so-called *threshold graph*. Let τ be any pairwise distance between points in P. Then the threshold graph G_τ contains an edge between two points iff their distance is at most τ.

Definition 19. *Let (P, d) be a metric space, and let $\tau \in \{d(x, y) \mid x, y \in P\}$.*

The threshold graph *$G_\tau = (P, E_\tau)$ is defined by $E_\tau = \{\{x, y\} \mid x, y \in P, d(x, y) \leq \tau\}$.*

Based on the threshold graph, we define a structure called *two-hop graph*. For that, we use the following definition which we reuse later on.

Definition 20. *For any graph $G = (V, E)$, we define $G^i = (V, E^i)$ recursively by $E^i = E^{i-1} \cup \{\{x, y\} \mid x, y \in V \wedge \exists z \in V : \{x, z\} \in E \wedge \{z, y\} \in E^{i-1}\}$.*

The *two-hop graph* is the graph $G_\tau^2 = (P, E_\tau^2)$, i.e., two points $x, y \in P$ are connected in G_τ^2 if they are connected in G_τ or if there is a third point to which both of them are connected in G_τ. Thus, the distance between x, y with $\{x, y\} \in E_\tau^2$ is at most 2τ.

Assume that τ is the optimum value, and that C is an optimum k-center clustering for P. Then for every point x in P, there is a center c in C such that x and c are adjacent in G_τ. Furthermore, if two points $x, y \in P$ have the same closest center in C, then x and y are adjacent in G_τ^2. Or, interpreted in a different way: Two points that are *not* adjacent in G_τ^2 can not be in the same cluster in any solution of maximum radius τ since there is no point which is in distance $\geq \tau$ of both of them. As a consequence, a 2-approximative solution for the k-center problem can be computed by finding a maximal independent set in G_τ^2. Before we formally prove this statement, recall the definition of independent sets.

Definition 21. *Let $G = (V, E)$ be an unweighted undirected graph. A set $I \subseteq V$ is an* independent set *if for all $u, v \in I$, $\{u, v\} \notin E$, i.e., no vertices in I are adjacent.*

When we talk about *maximal independent sets*, we mean that the set is *inclusionwise maximal*. Note that this is a different requirement than asking for a *maximum independent set*, which is an independent set of maximum cardinality. Computing maximum independent sets is NP-hard. In contrast to this, inclusionwise maximality means that no element can be added without destroying independence, and an independent set with this property can be found greedily.

Definition 22. *An independent set I in a graph $G = (V, E)$ is* maximal *if for every $u \in V \backslash I$, $I \cup \{u\}$ is not an independent set.*

maximal-independent-set$(G = (V, E))$

1. Set $I = \emptyset$ and $T = V$.
2. **Repeat**
3. Choose $u \in T$ arbitrarily and set $I = I \cup \{u\}$.
4. Set $T = T \backslash \{v \in T \mid v = u \vee \{u, v\} \in E\}$.
5. **Until** $T = \emptyset$
6. **Return** I

Observation 23. *Algorithm* ***maximal-independent-set***$(G = (V, E))$ *computes a maximal independent set in* G.

Proof. Observe that whenever a vertex u is added to I, all its neighbors are removed from T. Thus, since vertices are only added if they are still in T, the algorithm always maintains the property that I is independent. Furthermore, every vertex that is deleted from T has a neighbor in I and adding it to I thus destroys independence. This means that I is a maximal independent set. □

Now we prove that a maximal independent set in G_τ^2 provides a 2-approximation for the k-center problem.

Lemma 24. *Let* (P, d), k *be an input for the* k*-center problem, and assume that* τ *is the maximum radius of an optimal solution to the* k*-center problem on* (P, d), k. *Then*

1. *any independent set in* G_τ^2 *has cardinality at most* k, *and*
2. *any* maximal *independent set is a feasible solution to the* k*-center problem with maximum radius* 2τ.

Proof. Let I be any independent set in G_τ^2 and assume that $|I| > k$. Then I contains at least $k + 1$ points which have the property that no two of them can share the same closest center in the optimum solution (otherwise, they would be connected in G_τ^2). This is a contradiction since there are at most k clusters in the optimum solution. This proves 1.

Now assume that I is a maximal independent set and that there is a point x which is not within distance $\leq 2\tau$ of any point in I. Then x can be added to I without destroying independence. This is a contradiction to the assumption that I is maximal. This proves 2. □

Lemma 24 gives rise to the following algorithm.

k-`center-via-thresholding(`a finite metric space (P, d)`,` $k \in \mathbb{N}^{\geq 1}$`)`

1. Compute a list L of all pairwise distances $d(x, y)$ for $x, y \in P$, $x \neq y$
2. Set $\tau_{\min} := \max L$ and $C_{\min} := \{x\}$ for an arbitrary $x \in P$
3. **For** $\tau \in L$ **do**
4. Compute $C \subseteq P$ by calling `maximal-independent-set(`G_τ^2`)`
5. If $|C| \leq k$ and $\tau < \tau_{\min}$, set $\tau_{\min} = \tau$ and $C_{\min} = C$.
6. **Return** $C_{\min}$

An algorithm along this lines was first published by Hochbaum and Shmoys [HS85].

Theorem 25. *Algorithm* k`-center-via-thresholding(`(P, d)`,` k`)` *computes a 2-approximation for the k-center problem.*

Proof. The algorithm computes a maximal independent set in G_τ^2 for all possible choices of τ as a pairwise distance between points in P. Since the optimum solution is a pairwise distance between points in P, the algorithm in particular performs one iteration where τ is indeed the optimum value.

Let I^* be the solution computed by `maximal-independent-set(`G_τ^2`)` in the iteration where $\tau = r(C_k^{\text{opt}})$. By Observation 23, I^* is an independent set in G_τ^2. By 1 of Lemma 24, any independent set has cardinality $\leq k$, so in particular, $|I^*| \leq k$. This means that I^* is a feasible solution to the k-center problem on (P, d), k. Furthermore, by 2 of Lemma 24, the maximum radius of I^* is at most 2τ. Since τ is the optimum value, 2τ is at most twice the optimum value.

The solution that is finally output may not be this solution, but since we choose the solution with minimum τ that satisfies that the cardinality is at most k, this can only improve the final output. □

Compared to Gonzalez' algorithm, the thresholding algorithm for k-center seems complicated. However, we have now learned a lot about the structure of optimal solutions for k-center, and this knowledge will prove important for the next chapter. In fact, Gonzalez' algorithm is in some way a special case of the thresholding algorithm. By always choosing the point that is farthest away, it implicitly ensures that the point is independent of the previous points in G_τ^2 (for τ being the optimum value) as long as such a point exists. If no such point exists, then Gonzalez' algorithm already found a maximal independent set and simply adds additional centers until the cardinality of the center set is k. (We could similarly add additional centers to the solution of the thresholding algorithm if $|I^*| < k$). This can only improve the solution. So, Gonzalez' algorithm is one way to compute a maximal independent set (plus some extra centers), but the thresholding algorithm shows that there is a bit more flexibility in the choice of the centers than always choosing the one that is farthest away. Instead, any maximum independent set in G_τ^2 can be chosen as the center set.

2.6 A streaming algorithm for k-center

In the streaming model, we want to (approximately) solve problems in one pass over the input data.[1] There are different scenarios that motivate this model: In some applications, data is actually created at a speed that is so large that the data can not be stored. It is only available at the time of creation, and algorithms have to immediately process and filter it. This is for example the case for CERN's Large Hadron Collider. Another (and maybe more common) motivation is that nowadays, data is often of a size that may still fit on our hard drive, but not at all into the main memory of our computer.

The practical efficiency of algorithms that constantly read from the hard drive is so poor that streaming algorithms provide an immense speed-up because they only read every data point once from the hard drive, and this happens in blocks. The latter requirement is important, and it means that usually, we have to assume that the order of the points is given to us – randomizing or sorting the data requires additional passes over the data.

One-pass algorithms are not necessarily linear since could theoretically do extensive computations for each new data point. However, most of the time, this is not the case, and algorithms developed theoretically in the streaming model tend to be fast in practice as well. That has to do with the fact that they store very little data, and even doing much with this little amount of data is still quick. By saying *we only want to do one pass over the data*, we implicitly say that we do not want to store very much. We would like to store at most as much as fits into the main memory in order to avoid cache misses which would cause additional reads from the hard drive.

The aim when developing algorithms in the streaming model is to find algorithms that store as little as possible. How much do we need to store? That depends on the problem. For approximately solving k-clustering problems, we need at least $\Omega(k)$ space, since the desired output itself consists of k centers. The storage may also (need to) depend on the amount of data that the algorithm processes. A typical goal when designing streaming algorithms is that after reading n points of data, the storage requirement is a polynomial in $\log n$. Notice one detail in this formulation: we do not talk about the *size* of a *data set*, instead, we assume the view that the data is a *stream*, and we are interested in statements that hold after each data point, at which point the current *length* of the stream is the quantity which we compare our storage to.

For k-center, the streaming model is not well-defined without further considerations. Imagine an input to the k-center problem in the general metric case. It consists of a point set P and a metric d. If the points now arrive one by one, how do we get access to the metric? It seems unlikely that we would have the pairwise distances in the main memory if P itself is too large to fit into the memory.

[1] There exist streaming models where this requirement is relaxed and a small number of passes over the data is allowed. For some problems, that is unavoidable because they cannot be (approximately) solved in one pass over the data. However, for the problems considered in this lecture, we restrict the streaming model to one pass only.

So in this section, we assume that we have some sort of oracle access to the distances. That may seem absurd at first. However, if the data is for example from $\mathbb{R}^d$, then the oracle would just find the distance $d(x, y)$ between points $x, y \in \mathbb{R}^d$ by computing the Euclidean distance between them. The Euclidean space and distance may not be the modeling of our choice in a given application, but it is reasonable to assume that we have a way to compute a distance between two data points, instead of being given a finite metric space in form of a matrix or graph. Notice that this may require that we store more information on a point than just its number: For points in $\mathbb{R}^d$, storing a point needs $\Theta(d)$ space. For this reason, we will measure the storage of our k-clustering streaming algorithms in terms of the number of points that we store, not in terms of the actual storage required.

To summarize, a streaming algorithm for a k-clustering problem

- has access to a distance oracle,
- specifies the steps to be done after reading a new input point,
- stores very little (at least $\Omega(k)$ points and at most $\mathcal{O}(\text{poly}(k \cdot \log n))$, and
- is able to provide a solution for the k-clustering problem at any point in time.

– end of lecture 3 –

A streaming approximation algorithm for k-center

The following algorithm is due to Charikar, Chekuri, Feder and Motwani [CCFM04]. While not originally stated as a streaming algorithm, their algorithm can be formulated in the streaming model in a straightforward way as we see in the following.

Before the streaming algorithm is ready, the following routine is called to initialize some variables, in particular k, the number of clusters. This number is now fixed during all computations done on the following data stream.

`init`$(k \in \mathbb{N}^{\geq 1})$

1. Store the number k
2. Set $C := \emptyset$, $\ell = 0$ and init := `true`

Now the processing can start. We assume that the points come in some given order, and that for each point, the routine `update` is called (after the routine has finished for the previous point). This function has two modes: While reading the first k points, it simply stores the read points in C. Notice one detail: If a point arrives multiple times, the copies are of course ignored. Storing the first k distinct points is the only way to achieve any approximation since during this phase, the optimum solution (place the centers on the points) has value zero.

That changes when C contains k points and a new point x with $x \notin C$ arrives. This is the first time when any optimum solution has a non-zero cost. We can determine an optimum solution in this situation: It results from merging two points with minimum

distance. The function `update` now observes that the initialization phase has ended, adds x to C, removes one of the points of a closest pair from C and initializes a value ℓ to the smallest pairwise distance. Now $|C| \leq k$ is reestablished.

In the second mode, `update` ignores x if and only if there is a point in C which is at distance at most 2ℓ from x. Ignoring means that x already has a good center, so we do not need to do anything when processing x. If, however, $d(x, C) > 2\ell$, then ignoring x would increase the radius of the solution in an uncontrolled fashion. Thus, x is added to C. The function ends by calling `check`, which we describe below, and which ensures that $|C| \leq k$ is reestablished if necessary.

```
update(a point x from a metric space)
 1. If (init = true) then
 2.     If x ∉ C, add x to C
 3.     If |C| = k + 1 then
 4.         Set ℓ to the closest pairwise distance between points in C
 5.         Remove one point of a closest point pair from C
 6.         Set init := false
 7. Else
 8.     If d(x, C) > 2ℓ then
 9.         Add x to C
10.         check()
```

It remains to specify `check`. This function reduces the number of centers if necessary. It only has three lines: Check if $|C| > k$, and if so, replace C by a maximum independent set in $G_{2\ell}$ (Note the following subtlety: We compute the independent set in $G_{2\ell}$, the threshold graph where points are connected iff their distance is at most 2ℓ, and not in G_ℓ^2, the two-hop graph. If two points are connected in G_ℓ^2, then they are also connected in $G_{2\ell}$, but the reverse statement is not true.), and finally set $C = I$ and $\ell = 2\ell$. These steps are repeated until $|C| \leq k$. Notice that one iteration may not establish $|C| \leq k$ if the pairwise distances between the points in C are much larger than ℓ. However, once ℓ is at least half the smallest pairwise distance, the cardinality of C decreases. Notice that in order to implement `check` efficiently, one should compute the break point beforehand and directly increase ℓ sufficiently, but for the purpose of this analysis, considering the process as having several iterations is convenient.

```
check()
1. While |C| > k do
2.     Compute I ⊆ C by calling maximal-independent-set(G_2ℓ(C))
3.     Set C := I and ℓ = 2 · ℓ
```

For the sake of completeness, we also provide the following function which outputs the current solution. This function can be called at any time; if called while `update`

is still executed, it may however give a solution with $k+1$ centers. It should only be called between the executions of `update`.

```
solution()
  1. Return C
```

For the following analysis, we denote the set of the first $i \geq 1$ points in the stream by P_i. Since we now deal with multiple point sets, we use the more detailed notation

$$r(C, P_i) := \max_{x \in P_i} \min_{c \in C} d(x, c)$$

for the maximum radius of the solution $C \subset P_i$ with respect to the points in P_i.

Lemma 26. *For every $i \geq 1$, after processing the first i points in the stream with `update`, the following invariants hold:*

1. *The maximum radius $r(C, P_i)$ is at most 2ℓ.*
2. *All points in C have a pairwise distance of at least ℓ.*
3. *It holds $\ell \leq 4 \cdot r(C_k^{\text{opt}}, P_i)$, where C_k^{opt} is an optimal solution for the k-center problem on P_i. Thus, $\ell/4$ is a lower bound on the optimum value.*

Proof. As long as $i \leq k$, ℓ is still zero, which is a lower bound for the radius of any k-center solution, and also for all possible pairwise distances between the points. Furthermore, the radius $r(C, P_i)$ is also zero. Thus, all invariants are true.

For $i = k+1$, ℓ is set to the smallest distance between any pair of points among the first $k+1$ points. Notice that any k-center solution on P_{k+1} has to place two of the first $k+1$ points into the same cluster, and only the centers from P_{k+1} are available. Thus, the smallest pairwise distance is the optimum value for k-center on P_{k+1}, it holds $\ell = r(C_k^{\text{opt}})$. The point that is removed from C to reduce the number to k is a point who has a neighbor at distance ℓ (which remains in C), so $r(C, P_{k+1}) = \ell$ holds after removing it. Furthermore, all other points have a pairwise distance of at least ℓ. We conclude that all invariants are again true after the $(k+1)$th point.

Now fix any $i > k+1$ and assume that the invariants are true for $i-1$. If the ith point x_i is ignored by `update`, meaning that line 8 produced `false`, then the new point is within distance 2ℓ of a point in C. Since $r(C, P_{i-1}) \leq 2\ell$, and $d(x_i, C) \leq 2\ell$, $r(C, P_i) \leq 2\ell$. No new point is added to C, so `check()` is not called, and the optimum value can only increase, so all invariants are again true.

Now if $d(x_i, C) > 2\ell$, then x_i is added to C. This only affects invariant 2., and this invariant stays true: the point is added to C because it is at distance $> 2\ell > \ell$ of any point in C. So all invariants are true before `check()` is called.

Now we consider what happens during the execution of `check`. We know that the invariants are true when we enter the function. Now we show that the invariants are true after each iteration of the `while`-loop. So assume that we are in line 1 of `check` and all invariants are true. If $|C| \leq k$, nothing happens and we are done. If $|C| > k$, we observe that because of invariant 2., this means that we have at least $k+1$ points

which have a pairwise distance of at least ℓ (in fact we have exactly $k+1$ points because `check` is called after every point insertion). Since any optimum solution has to place two of them into the same cluster, $r(C_k^{\text{opt}}, P_i) \geq \ell/2$. Line 2 of `check` replaces C by an independent set I in $G_{2\ell}$. All points in I have a pairwise distance of more than 2ℓ. Furthermore, every point in $C \backslash I$ has a distance of at most 2ℓ to its closest point in I. Now let $x \in P_i \backslash C$ be a point that has been deleted or removed at an earlier point in time. By invariant 1., $d(x, C) \leq 2 \cdot \ell$. By the triangle inequality, $d(x, I) \leq 4 \cdot \ell$ since every point in C has distance at most 2ℓ to I. We have now seen:

- for any optimum solution C_k^{opt}, $r(C_k^{\text{opt}}, P_i) \geq \ell/2$,
- all points in I have a pairwise distance of more than 2ℓ, and
- the maximum radius of the solution I is bounded by $r(I, P_i) \leq 4\ell$.

This means that after line 3, where C is set to I and ℓ is replaced by 2ℓ, all invariants are again true. □

We have already shown our main result, the following theorem is a corollary: Invariants 1. and 3. imply that after point i is fully processed, we have $r(C, P_i) \leq 2 \cdot \ell \leq 8 \cdot r(C_k^{\text{opt}})$.

Theorem 27. *For all $i \geq 1$, if the first i points have been processed by* `update`*, then the solution $C =$* `solution()` *is an 8-approximation for the k-center problem on P_i. The algorithm stores $\Theta(k)$ points.*

2.7 The k-center problem with outliers

[In winter semester 2018/2019, the section on k-center with outliers is excluded from possible questions in the oral exam.]

In this section, we apply our knowledge about the structure of the k-center problem to develop an approximation algorithm for a variant of the k-center problem.

Definition 28. *Let (P, d) be a metric space, let $k, z \in \mathbb{N}^{\geq 0}$ be numbers. The* k-center problem with outliers *is to compute a set $C \subseteq P$ with $|C| = k$ and a set of outliers $Z \subseteq P$ with $|Z| \leq z$ such that*

$$\max_{x \in P \backslash Z} \min_{c \in C} d(x, c)$$

is minimized.

The k-center problem with outliers allows z points to be marked as *outliers*. They are ignored when computing the maximum radius of a solution. Why do we say that this adds a constraint to the problem? That is because we can also view it differently: The k-center problem with outliers asks for $k' = k + z$ centers, but z of these centers may only serve themselves. Thus, we have the additional constraint that z centers can not serve other points.

– end of lecture 4 –

The k-center problem with outliers can be 2-approximated [CGK16]. However, this result is a special case of an algorithm for a much more general case (the non-uniform k-center problem), and does not fit too well here. Instead, we will consider an easier variation which is based on the algorithm by Hochbaum and Shmoys and gives a 3-approximation for the problem.

The main algorithm is very similar to the thresholding algorithm for the standard k-center problem:

`thresholding-with-constraints(`a finite metric space (P, d)**`,`** and $k, z \in \mathbb{N}^{\geq 1}$**`)`**

1. Compute a list L of all pairwise distances $d(x, y)$ for $x, y \in P$, $x \neq y$
2. Set $\tau_{\min} := \max L$, $Z_{\min} = \emptyset$ and $C_{\min} := \{x\}$ for an arbitrary $x \in P$
3. **For** $\tau \in L$ **do**
4. Compute (C, Z) with $C, Z \subseteq P$, $|Z| \leq z$
   by calling `ind-set-outlier(`G`,`F`,`z`,`τ`)`
5. If $|C| \leq k$ and $\tau < \tau_{\min}$, set $\tau_{\min} = \tau$, $Z_{\min} = Z$ and $C_{\min} = C$.
6. **Return** $C_{\min}$ and $Z_{\min}$

Aside from the fact that we have to keep track of the outlier set $Z_{\min}$ in addition to I, the main change is that we use a different subroutine to compute the actual center set. The idea of the algorithm is to try to find the center of an optimal cluster (or a similarly good point) by picking a point that has many points close to it. Since we have a guess τ for the optimum value available, close just means that the points are within distance τ. In the subroutine, we store all yet uncovered points in a set T (in the beginning, $T = P$).

For any $x \in P$, we then count the number of uncovered points that x has in its neighborhood. More precisely, we determine the cardinality of the set

$$N(x) = \{z \in T \mid d(x, y) \leq \tau\}$$

for all points in P. We also call $N(x)$ the *direct neighborhood* of x. While the direct neighborhood is needed for determining the next center, we need the *extended neighborhood* of a point $x \in P$ for deciding which points shall now be considered as covered. The extended neighborhood is given by

$$E(x) = \{z \in T \mid d(x, y) \leq 3\tau\}$$

Here is the subroutine that we use:

ind-set-outlier$((P,d),\ z,\ \tau)$

1. Set $I = \emptyset$ and $T = P$
2. **Repeat**
3. Choose $u \in P$ with $|N(u)| = \max\{|N(x)| \mid x \in P\}$ and set $I = I \cup \{u\}$.
4. Remove all vertices in $E(u) \cap T$ from T
5. **Until** $|T| \leq z$
6. **Return** (I, T)

Routine `ind-set-outlier` starts by copying P to T, i.e., in the beginning, all points are uncovered. It then repeatedly chooses a center and removes points from T which are now considered covered. The next center u is chosen based on the size of the direct neighborhood. One of the points with the largest direct neighborhood is selected to be u, and then u is added to I. The next step is to compute the extended neighborhood $E(u)$ and to delete all points in $E(u)$ from T. Note that $E(u)$ is not deleted from P; we allow all points in P to be centers. Side remark: This means that the computed set I will not be an independent set in $G_{3\tau}$ (but we have a new argument to replace our previous argumentation, so it is not a problem). It is an interesting exercise to prove that I will be, in fact, an independent set in $G_{2\tau}$.

After removing $E(u)$, the routine checks whether the number of points T is still larger than z and, if so, repeats the process. Once $|T|$ has decreased sufficiently, the routine returns (I, T). Thus by definition, Routine `ind-set-outlier` always computes a solution with at most z outliers. It may, however, compute more than k centers. In the following lemma, we show that this can not happen if the subroutine is called with $\tau \geq r(C_k^{\text{opt}})$.

Lemma 29. *If τ is at least the maximum radius of an optimal solution for the k-center problem with outliers and forbidden centers on (P,d), z, $F \subset P$, k, then* ***ind-set-outlier****$((P,d),\ F, z, \tau)$ computes (I,T) with $|I| \leq k$ and $|T| \leq z$.*

Proof. We relate the output of the algorithm to a fixed optimal solution. An optimal solution consists of a center set C^{opt} and a set of outliers Z^{opt}. We are interested in a partitioning induced by C^{opt}, and we name this partitioning $\mathcal{A}$ (so it is a set of k disjoint subsets of P). The union of the sets in $\mathcal{A}$ is $P \backslash Z$.

Some more notation: For the purpose of this proof, we name the center chosen in the ith iteration of `ind-set-outlier` by u_i. Its direct neighborhood is abbreviated by $N_i := N(u_i)$ and its extended neighborhood is $E_i := E(u_i)$. Furthermore, we let T_i be the set T at the *beginning* of iteration i.

Our goal is now to show that routine `ind-set-outlier` covers at least $\sum_{A \in \mathcal{A}} |A|$ points with the first k centers. That then means that the set of outliers contains less than $|P| - \sum_{A \in \mathcal{A}} |A| \leq z$ points after the kth iteration, i.e., the routine stops and returns a feasible solution.

We make a case distinction. First, we consider all clusters A in $\mathcal{A}$ with the property that at least one point from A was in the direct neighborhood of a center that

`ind-set-outlier` chose in the first k iterations. We call the set of all these clusters $\mathcal{G} \subseteq \mathcal{A}$, and the set of the remaining clusters $\mathcal{B} = \mathcal{A} \backslash \mathcal{G}$. For every cluster $A \in \mathcal{G}$, we let $i(A) \in [k]$ be the smallest number for which $A \cap N_{i(A)} \neq \emptyset$.

What does this mean? Let $x \in A \cap N_i$ be the point that is in the non-empty intersection (set $i = i(A)$ for this paragraph). Then every point in A has a distance of at most 2τ to x, since they are in the same optimum cluster as x. But the distance between x and u_i is at most τ, since $x \in N_i$ as well. This means that every point in A has a distance of at most 3τ to u_i, implying that $(A \cap T_i) \subseteq E_i$. Since all points in E_i are deleted in iteration i, this means that $A \cap T_i$ is deleted from T. Notice that this also means that the $i \in [k]$ with $A \cap N_i \neq \emptyset$ is unique: After the first iteration where this happens, no points from A remain in T.

What about the points from A that were deleted from T *before* iteration i? Since iteration i is the first iteration where a point from A was in the direct neighborhood of the chosen center, points there were previously deleted were *not* in the direct neighborhood, but only in the extended neighborhood.

Let $I_G = \{i \in [k] \mid \exists A \in \mathcal{G} : i = i(A)\}$ and $I_B = [k] \backslash I_G$. Then we have just shown that

$$\bigcup_{A \in \mathcal{G}} A \subseteq \left(\bigcup_{i \in I_G} E_i \cup \bigcup_{i \in I_B} E_i \backslash N_i \right).$$

Notice that $|I_G| \leq |\mathcal{G}|$: while multiple clusters from $\mathcal{G}$ may be deleted in the same iteration, there is only one fixed iteration that is added to I_G for each $A \in \mathcal{G}$. Thus, also $|I_B| \geq |\mathcal{B}|$ is true.

Now we know that all points in $\bigcup_{A \in \mathcal{G}} A$ are removed and are not in T after iteration k. What points can still be in T? Let $A \in \mathcal{B}$ be one of the optimum clusters we have not considered so far. It may be that some of the points in A are removed during the k iterations, which is only good for us. Indeed, we can strengthen our above statement to:

$$\left(\bigcup_{A \in \mathcal{G}} A \cup \bigcup_{A \in \mathcal{B}} A \backslash r(A) \right) \subseteq \left(\bigcup_{i \in I_G} E_i \cup \bigcup_{i \in I_B} E_i \backslash N_i \right),$$

where $r(A)$ is the subset of A containing all points still remaining after iteration k. This means that This implies that

$$\left(\sum_{A \in \mathcal{G}} |A| + \sum_{A \in \mathcal{B}} (|A| - |r(A)|) \right) \leq \left(\sum_{i \in I_G} |E_i| + \sum_{i \in I_B} |E_i \backslash N_i| \right),$$

and now have to show that $\sum_{A \in \mathcal{B}} |r(A)|$ is not too large.

Since all points in $r(A)$ are not removed during the first k iterations, $|A \cap T_i| \geq |r(A)|$ holds for all $i \in [k]$. In particular, the center $c(A)$ of A (which is a point in P and thus always a possible center) has a neighborhood of size at least $|r(A)|$. Since we always choose a point with a neighborhood of maximum size, $|N_i| \geq |r(A)|$ is always true: for all $i \in [k]$ and all $A \in \mathcal{B}$. Together with the fact that $|I_B| \geq |\mathcal{B}|$, this implies

$$\sum_{i \in I_B} |N_i| \geq \sum_{A \in \mathcal{B}} |r(A)|.$$

In words, this inequality says the following: What we remove (in addition to $\bigcup_{i\in I_G} E_i \cup \bigcup_{i\in I_B} E_i\backslash N_i$) is at least as much as what we should remove (in addition). Formally, we have shown that

$$\begin{aligned}\sum_{A\in\mathcal{A}} |A| &= \left(\sum_{A\in\mathcal{G}} |A| + \sum_{A\in\mathcal{B}} (|A| - |r(A)|)\right) + \sum_{A\in\mathcal{B}} |r(A)| \\ &\leq \left(\sum_{i\in I_G} |E_i| + \sum_{i\in I_B} |E_i\backslash N_i|\right) + \sum_{i\in I_B} |N_i| \\ &= \sum_{i=1}^{k} |E_i|,\end{aligned}$$

and that completes our proof. □

Lemma 29 contains the main technical step to prove the following result by Charikar et. al. [CKMN01].

Theorem 30. *Algorithm* `thresholding-with-constraints`$((P,d),\ F,\ k,\ z)$ *computes a 3-approximation for the k-center problem with outliers and forbidden centers.*

Proof. Similarly to Observation 18, we observe that only the pairwise distances present in (P,d) are possible candidates for the value of an optimal solution to the k-center problem with outliers and forbidden centers on (P,d), F, k, z. Since the algorithm `thresholding-with-constraints` iterates through all of them, it performs one iteration of `ind-set-outlier` where τ is the optimum value. By Lemma 29, the center set C computed by this call satisfies $|C| \leq k$. By line 3 of `ind-set-outlier`, C only contains points from $P\backslash F$. By line 6 of `ind-set-outlier`, we know that $|T| \leq z$. Furthermore, all points not in T have been removed because they are neighbors of a point in C in G_τ^3, and are thus within distance 3τ of a point in C. Since τ is the optimum value, C and T are a 3-approximative solution for the k-center problem with outliers and forbidden centers. The solution that is finally chosen is either this solution or one that is only better. □

2.8 The fair k-center problem

Adding constraints to a clustering problem can mean many things. We have seen one constraint (clustering with outliers) where we restrict the number of points that can be assigned to some centers (a set of z centers can only be assigned at most one point, namely themselves). Another classical variant of the k-center problem would be to add upper or lower capacities to all centers. For example, the capacitated k-center problem asks for a set of centers C and an assignment of points to centers such that the number of points assigned to a center c does not violate its capacity $u(c)$ (notice the difference to the outlier version: The capacity is here given for each center c individually, which is different from saying that there should be a center with at most a certain number of points). The lower-bounded k-center problem restricts the number of assigned points

from below, not above, by demanding that every center gets at least as many point as its lower bound $\ell(c)$ states. Both these problems can be approximated; indeed, the lower-bounded k-center problem allows for a 3-approximation [AS16], and the capacitated k-center problem for a 6-approximation [KS00] (matching lower bounds are not known).

In this section, we want to study a problem that is loosely related to k-center with upper and lower capacities, but has a very modern motivation. This is the *fair k-center problem* as proposed by Chierichetti, Kumar, Lattanzi and Vassilvitskii [CKLV17].

The model for fair k-center strives to find a clustering where all clusters have the same composition with respect to some sensitive attribute as the whole point set. As an example, consider assigning children to kindergartens in a way that ensures that the number of boys and the number of girls is roughly equal.

The difficulty of the fair k-center problem depends highly on the exact mathematical formulation. For the purpose of this lecture, we will restrict ourselves to binary attributes as in the example of boys and girls. This case is significantly easier to model and to approximate than the more general formulation where the number of different possible attribute values is larger than two.

Definition 31. *Let (P, d) be a metric space, and let $a : P \to \{0, 1\}$ be a mapping. For any subset $S \subseteq P$ we use the notations S_0 and S_1 to denote the sets of points with attribute value 0 and 1, respectively, i.e.,*

$$S^j = \{x \in S \mid a(x) = j\}$$

for $j \in \{0, 1\}$. Then we call

$$\text{ratio}(S, j) = \frac{|S^j|}{|S|}$$

the ratio *of attribute value j in S, and we say that S is* exactly balanced *if*

$$\text{ratio}(S, 0) = \text{ratio}(P, 0).$$

*[This implies $\text{ratio}(S, 1) = \text{ratio}(P, 1)$ since we only have two attribute values.] For convenience, we assume that at least half of the points have attribute value 1 (otherwise, we switch the naming). We say that S is (ℓ, u)-*balanced *for $\ell \in [0, \text{ratio}(P, 0)]$ and $u \in [\text{ratio}(P, 0), 1]$ if*

$$\ell \leq \text{ratio}(S, 0) \leq u.$$

Achieving exactly balanced clusters according to Definition 31 can be difficult. Imagine for example a point set P with 49 points with attribute value 0 and 51 points with attribute value 49. Then no true subset of P is exactly balanced according to Definition 31 because a balanced cluster has to have $\text{ratio}(C, 0) = 49/100$, and the only subset of P satisfying this constraint is P itself. Choosing ℓ and u away from $49/100$ allows to split P into multiple clusters. For arbitrary ℓ and u, the problem can still be complicated even for two attribute values. We define the following special case as our variant of the fair k-center problem.

Definition 32. *Let (P, d) be a metric space, let $a : P \to \{0, 1\}$ be a mapping and let $k, t \in \mathbb{N}^{\geq 1}$ be numbers (where t is such that $\frac{1}{t} \leq \text{ratio}(P, 0)$). The* fair k-center problem *is to compute a set $C = \{c_1, \ldots, c_k\} \subseteq P$ with $|C| = k$ and an assignment $c : P \to C$ such that the induced clusters $C_1, \ldots, C_k$ (with $C_i = \{x \in P \mid c(x) = c_i\}$) are $(\frac{1}{t}, 1)$-balanced and*

$$r(C, c) := \max_{x \in P} d(x, c(x))$$

is minimized.

– end of lecture 5 –

Definition 32 demands a solution where the ratio of the points with attribute value 0 is at least $1/t$, i.e., in any cluster we have that for each point of value 0, there are at most $t - 1$ points of attribute value 1. It allows any t with $(1/t) \leq \text{ratio}(P, 0)$, so the only restriction here is that we do not allow irrational numbers. Furthermore, we drop the upper bound u and set it to 1.

Lemma 33. *Assume that the center set C^* and the assignment $c^* : C \to P$ form an optimal solution to the fair k-center problem on (P, d, a, k, t). Then there is an assignment $c' : P^1 \to P^0$ such that*

1. *for all $x \in P^0$, $|\{y \in P^1 \mid c'(y) = x)\}| \leq t - 1$, and*
2. *for all $y \in P^1$, $d(y, c'(y)) \leq 2r(C^*, c^*)$.*

Proof. We construct c' by considering each cluster in the optimum solution individually. So let S be a cluster in the optimum solution. We know that the solution is $(\frac{1}{t}, 1)$-balanced, so in particular, $\text{ratio}(S, 0) \geq 1/t$. By definition, this means that

$$\frac{|S^0|}{|S^0| + |S^1|} \geq \frac{1}{t} \Leftrightarrow |S^1| \leq (t - 1)|S^0|.$$

We can thus split S^1 into $|S^0|$ groups of at most $t - 1$ points each, and then assign each group to a unique point in S^0. This satisfies requirement 1. For requirement 2., observe that all points in S have a distance of at most twice the maximum radius. □

Lemma 33 ensures us that there is a good clustering into $|P^0|$ clusters, where the points in P^0 are the centers. This is a crucial observation since it is much easier to find a clustering with this structure than to find a fair k-clustering directly. Using the finer $|P^0|$-clustering, we can then later construct a fair k-clustering.

If we already know that the points in P^0 are supposed to be the centers, all that is left to do is to compute the assignment of points with attribute value 1 to these centers, in a way that ensures sufficient balance. Finding this assignment is (basically) the task in Problem 3 on Problem Set 5, and we defer the solution to the tutorial (proof will later be added).

Lemma 34. *Let (P, d) be a metric space, let $a : P \to \{0, 1\}$ be a mapping and let $k, t \in \mathbb{N}^{\geq 1}$ be numbers (where t is such that $\frac{1}{t} \leq \text{ratio}(P, 0)$). Assume that a set of centers C is given, and an assignment $a : P^0 \to C$ of the points with attribute value 0 to these centers is given as well. Extending c to P^1 such that*

- *all induced clusters are* $(\frac{1}{t}, 1)$*-balanced, and*
- *the maximum radius of* (C, c) *is minimal*

is possible in polynomial time.

Lemma 34 says that if C and $c : P^0 \to C$ are fixed, then extending c optimally is possible in polynomial time. In the tutorial, this lemma is used in a scenario where the points in P^0 are already assigned; here, we use it in a scenario where the points in P^0 are the centers (and thus implicitly assigned to themselves). After using Lemma 34 to compute an assignment of then points in P^1 to the points in P^0, we use a k-center 2-approximation of our choice to cluster the points in P^0. And then, all we have to do is to combine the two clusterings.

`fair-k-center`$((P,d),\ a : P \to \{0,1\},\ k, t \in \mathbb{N}^{\geq 1})$

1. Check that $\text{ratio}(P, 0) \leq 0.5$ and that $\frac{1}{t} \leq \text{ratio}(P, 0)$
2. Find an optimal assignment $c' : P^1 \to P^0$ of the points in P^1 to P^0 under the condition that $(c')^{-1}(x) \leq t - 1$ for all $x \in P^0$ (Lemma 34)
3. Call `farthest-first-traversal`$((P^0, d),k)$ to compute a center set C
4. **For all** $x \in P^0$
5. Set $c(x) = \arg\min_{c \in C} d(x, c)$
6. **For all** $y \in P^1$
7. Set $c(y) = c(c'(y))$
8. **Return** (C, c)

Theorem 35. *Let* (P, d) *be a metric space,* $a : P \to \{0, 1\}$ *be a mapping and* $t \in \mathbb{N}^{\geq 1}$ *such that* $\frac{1}{t} \leq \text{ratio}(P, 0)$*. Then Algorithm* ***`fair-k-center`*** *computes a* 4*-approximation to the fair* k*-center problem on* $((P, d), a, t)$*.*

Proof. By Lemma 33, we know that it is possible to assign the points in P^1 to the points in P^0 such that the maximum distance between any point $y \in P^1$ to its assigned point $c'(y) \in P^0$ is at most $2r(C^*, c^*)$, where (C^*, c^*) is an optimal solution, and such that for any point $y \in P^0$, $(c')^{-1}(y) \leq t - 1$. By Lemma 34, we know that we can find an optimal assignment of the points in P^1 to the points in P^0 under the condition that every point in P^0 gets at most $t - 1$ points; such an optimal assignment then has a radius of at most $2r(C^*, c^*)$ as well. So the assignment c' computed in line 2. satisfies that $d(y, c'(y)) \leq 2r(C^*, c^*)$ for all $y \in P^1$.

Next, the algorithm clusters the points in P^0 by using a 2-approximation for the k-center problem. The optimum solution to the unconstrained k-center problem on $(P^0, 0)$ can only be cheaper than the solution to the fair k-center problem on the full point set. Thus, the maximum distance between any point $x \in P^0$ and its closest center $c(x) = \arg\min_{c \in C} d(x, c)$ in the computed set C is at most $2r(C^*, c^*)$.

Finally, the algorithm sets the center of any $y \in P^1$ to $c(c'(y))$. By the triangle inequality,

$$d(y, c(c'(y))) \leq d(y, c'(y)) + d(c'(y), c(c'(y))) \leq 2r(C^*, c^*) + 2r(C^*, c^*) \leq 4r(C^*, c^*),$$

so every point in P^1 is at distance at most $4r(C^*, c^*)$ of is final center, which proves the theorem. □

Chapter 3

The exciting world of k-means

In this chapter, we consider the probably most known clustering problem: The k-means problem. Even more famous than this problem is a heuristic for it (Lloyd's algorithm) which we discuss later and which is commonly often referred to as *the* k-means algorithm. We will argue that nowadays, there are improvements of this algorithm that should be considered instead of implementing the heuristic in its vanilla version.

Furthermore, we will see how the k-means problem can be approximated in the streaming model, i.e., by one-pass algorithms. The resulting combination of a practical approximation algorithm and streaming routines lead to very powerful practical algorithms for the k-means problem.

3.1 The (Euclidean) k-means problem

Here is a formal definition of the k-means problem:

Definition 36. *Let $P \subset \mathbb{R}^d$ be a finite set of points from the Euclidean space and $k \in \mathbb{N}^{\geq 1}$ be a number. The (Euclidean)* k-means problem *is to compute a set $C \subset \mathbb{R}^d$ with $|C| = k$ that minimizes*

$$\text{dist}^2(P, C) := \sum_{x \in P} \min_{c \in C} ||x - c||^2.$$

In the following we also use the abbreviation $\text{dist}^2(P, z) := \text{dist}^2(P, \{z\})$ for denoting the sum of the squared distances of all points in P to a single point z. Furthermore, we sometimes consider *weighted* k-means. Then the input point set comes together with a weight function $w : P \to \mathbb{R}$, and the objective to minimize changes to

$$\text{dist}^2(P, w, C) := \sum_{x \in P} w(x) \cdot \min_{c \in C} ||x - c||^2.$$

If the weights are integral, we can think of the weights as multiplicities.

The k-means problem is NP-hard, and it is NP-hard to approximate it better than 1.0013 (see [ACKS15, LSW17]). There are at least three papers proposing constant-factor approximation algorithms for the k-means problem, the best of which achieves an approximation ratio of 6.357 (see [ANSW17]). All of these are based on algorithms for the k-median problem, the close sibling of the k-means problem:

Definition 37. *Let (P, d) be a metric space and $k \in \mathbb{N}^{\geq 1}$ be a number. The* k-median problem *is to compute a set $C \subseteq P$ with $|C| = k$ that minimizes*

$$\sum_{x \in P} \min_{c \in C} d(x, c).$$

Compared to the k-median problem, we notice two major changes. First of all, the metric space is the Euclidean space. That is a restriction: Instead of an arbitrary space of which we only know that it satisfies the three axioms in Definition 1, we know exactly how the distances are computed (see Fact reffact:euclideannormisanorm).

However, the k-means problem is *not* a special case of the k-median problem. On the contrary: Firstly, the objective function considers squared Euclidean distances. These are not metric; so while the k-means problem is based on a specific metric (Euclidean distances), its cost is not actually metric. Indeed, the cost that a point incurs does not satisfy the triangle inequality. This is a major challenge when adapting approximation algorithms for k-median to k-means. Often, the way to resolve it (at the cost of a worse bound) is to use the fact that squared distances satisfy the following *relaxed triangle inequality*.

Fact 38. *Let $x, y, z \in \mathbb{R}^d$ be points. It holds that*

$$||x - y||^2 \leq 2||x - z||^2 + 2||z - y||^2.$$

Secondly, notice that the center set C can be chosen from *all* of $\mathbb{R}^d$ in the case of the k-means problem, while in the k-median problem, it is (usually) restricted to the input point set. For the k-median problem, an optimal solution can be computed in time $\binom{n}{k} \in \Theta(n^k)$ by testing all possible choices of k centers from P. For the k-means problem, it is not straightforward to obtain a similar result. It is possible test all partitionings of P into k subsets and then finding the best center for each subset (we see how to do this in the next section), but the running time of this is $\Theta(k^n)$, which is in general much worse than $\Theta(n^k)$. Inaba, Katoh and Imai [IKI94] show how to optimally solve the k-means problem in time $n^{\mathcal{O}(dk)}$. This algorithm is based on enumerating Voronoi diagrams.

3.2 Lloyd's algorithm

The extremely popular heuristic often called *the k-means algorithm* which we discuss in the following is usually attributed to a note by Lloyd in 1957 which was later published in [Llo82], but it has also (independently) been discovered by Steinhaus [Ste56]. The interested reader is referred to [Boc07] for a historical overview.

The heuristic is based on the following extremely helpful observation which is usually considered folklore. The following proof is similar to the proof in [KMN+04].

Fact 39. *Let $P \subset \mathbb{R}^d$ be a finite point set and let $\mu(P) := \frac{1}{|P|}\sum_{x\in P} x$ be its centroid. Then, every point $z \in \mathbb{R}^d$ satisfies*

$$\sum_{x\in P} \|x - z\|^2 = \sum_{x\in P} \|x - \mu(P)\|^2 + |P| \cdot \|\mu(P) - z\|^2.$$

Notice this is the same as saying that $\text{dist}^2(P, z) = \text{dist}^2(P, \mu(P)) + |P| \cdot ||\mu(P) - z||^2$.

Proof. We abbreviate $\mu = \mu(P)$ during the proof. Notice that for all points $x, y \in \mathbb{R}^d$, it holds that $||x+y||^2 = \langle x+y, x+y\rangle = \langle x, x\rangle + 2\langle x, y\rangle + \langle y, y\rangle = ||x||^2 + 2\langle x, y\rangle + ||y||^2$. This implies that $||x-z||^2 = ||x-\mu+\mu-z||^2 = ||x-\mu||^2 + 2\langle \mu - z, x-\mu\rangle + ||\mu-z||^2$. Furthermore, it holds that

$$\sum_{x\in P}(x-\mu) = \left(\sum_{x\in P} x\right) - |P|\cdot\mu = |P|\cdot\mu - |P|\cdot\mu = 0.$$

Combining both statements implies that

$$\begin{aligned}\sum_{x\in P} ||x-z||^2 &= \sum_{x\in P} \|x-\mu\|^2 + 2(\mu - z)^T \sum_{x\in P}(x-\mu) + |P|\cdot\|z-\mu\|^2 \\ &= \sum_{x\in P} \|x-\mu\|^2 + |P|\cdot\|z-\mu\|^2.\end{aligned}$$

□

Fact 39 has a lot of consequences. The most obvious consequence is that the *centroid* of a point set is the best 1-means solution: The best 1-means solution is the point z that minimizes $\sum_{x\in P} ||x-z||^2$. By Fact 39, this sum has a fixed part, $\|x-\mu(P)\|^2$, and only the additional term $|P|\cdot\|z-\mu(P)\|^2$ actually depends on z. It becomes zero iff $z = \mu(P)$. This also explains how iterating through all partitionings of P leads to an (exponential time) algorithm that optimally solves the k-means problem. In any optimal solution, the center of any cluster has to be chosen as the centroid of the cluster. Thus, the optimal solution can be found by iterating through all partitionings and optimally computing the centers for the subsets.

Another consequence of Fact 39 is that it gives rise to Lloyd's algorithm

Lloyds-algorithm($P \subset \mathbb{R}^d$, $k \in \mathbb{N}^{\geq 1}$)

1. Choose a set of k centers $C \subset \mathbb{R}^d$ arbitrarily
2. **Repeat**
3. Set $C' = \emptyset$
4. Compute a clustering $C_1, \ldots, C_k$ induced by C
5. **For each** $i \in [k]$ **do**
6. **If** $|C_i| > 0$
7. Add $\mu(C_i)$ to C'
8. Replace C by C'
9. **Until** a stopping criterion is met
10. **Return** (C)

Algorithm `Lloyds-algorithm` can be implemented in different ways. Firstly, the initial choice of centers is not specified. A common method is to choose k points from P uniformly at random. The main improvement of Lloyd's algorithm that we will see later on is to replace this step by a better initialization method. Secondly, the algorithm as stated above lets clusters run empty. A practical improvement would be to add a completely new center to C' whenever a cluster ran empty. If doing this, we again have to specify how the new center is chosen (for example, uniformly at random from P, or along the lines of the improved initialization that we see later), and have to make sure that this can not make the algorithm cycle (see Observation 40 below). Thirdly, the stopping criterion is not stated. There are two common choices: Either stopping after a fixed number of iterations I, or stopping when the solution quality does not decrease 'significantly' anymore, i.e., when in an iteration, the new solution is less than T better than the previous solution for some fixed T. Both methods need the user to specify a parameter; specifying the number of iterations has the advantage that the running time of the algorithm is bounded by $\mathcal{O}(Indk)$. Bounding the minimum improvement should only be done based on some knowledge of the magnitude of the optimum cost. It is also possible to run Lloyd's algorithm until convergence.

Observation 40. *After finitely many steps of Lloyd's algorithm, $C' = C$ holds before line 8, i.e., the algorithm converges.*

Proof. This is true because every iteration where C' differs from C improves the solution. Say we start with C and assign every point to its closest center. For a given center set, that is the best possible assignment. Now if C does not exactly contain the centroids of the clusters, then the cost of the solution is improved when the centers are replaced by the centroids because of Fact 39.

Further notice that after the first iteration, the algorithm only considers center sets that are sets of centroids of a partitioning. Since every change strictly improves the cost, we can not encounter the same centroid set again. Thus, the algorithm has to stop after at most $\mathcal{O}(k^n)$ iterations. □

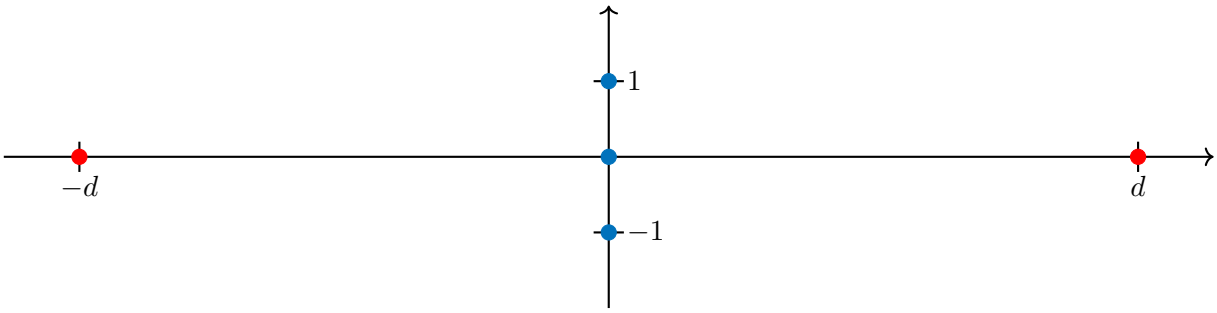


Figure 3.1: An example with five points where the blue points form a local but not global optimum for $k = 3$. (Figure copied from [Sch14].)

Lloyd's algorithm is a *local search heuristic*: Given a solution for the problem, it does a local improvement step, and when run to convergence, it produces a *local optimum*. However, the quality of such a local optimum can be bad. Consider Figure 3.1: Once Lloyd's algorithm chooses the blue points as the centers, the solution is locally optimal, but worse than the optimum solution by a factor of $\Omega(d^2)$.

Furthermore, even though Lloyd's algorithm often reportedly converges quickly in many practical applications, its running time is not bounded by a polynomial in the input size in general. Indeed, Vattani [Vat11] proposes a family of instances in $\mathbb{R}^2$ where Lloyd's algorithm needs exponentially many iterations. However, in practice, Lloyd's algorithm is not necessarily run until convergence as we discussed above. Also, it has been shown that the *smoothed complexity* of Lloyd's algorithm is indeed polynomial. Smoothed analysis provides a model for analyzing algorithms under the assumption that inputs are a bit randomized and thus not completely adversarial. Details on the smoothed analysis of Lloyd's algorithm are discussed in the lecture *Randomized Algorithms and Probabilistic Analysis*, see [RS18].

3.3 The k-means++ algorithm

The k-means++ algorithm is a combination of a powerful seeding technique called D^2-*sampling* and Lloyd's algorithm. It was proposed by Arthur and Vassilvitskii [AV07] and has had a major impact on practical and theoretical studies of the k-means problem. It is named after Lloyd's algorithm, more precisely, after its alternative name (k-means algorithm).

From a theoretical point of view, D^2-sampling is the main contribution of Arthur and Vassilvitskii's work. Indeed, D^2-sampling alone provides an (expected) approximative solution as we will see in this section. From a theoretical point of view, it is not clear that Lloyd's algorithm improves upon the guarantee of D^2-sampling in a significant manner.

On the other hand, from a practical point of view, k-means++ is often considered as a 'variant' of Lloyd's algorithm, which just has a very good initialization. So from

that perspective, running Lloyd's algorithm is crucial; and indeed, when performing experiments, it is obvious that the improvement by refining the solution with Lloyd's algorithm is significant.

So k-means++ should really be seen as a combination of sampling and Lloyd's algorithm, which in its entirety is a very good algorithm. At least in practical experiments, it produces much better solutions than each of the two algorithms individually.

D^2-sampling can intuitively be seen as a probabilistic variant of Gonzalez' algorithm. Recall that for Gonzalez' algorithm for k-center, we pick the first point arbitrarily and then subsequently pick the point that is farthest away from all previously chosen centers. For objectives like k-median and k-means, we have the additional complication that we have to take into accounts that multiple points cost more than small amounts. Here is a small picture to illustrate this fact:

Each point has a name depicted below it, and a weight depicted above it. Assume we were to solve k-center on this instance, run Gonzalez' and pick a as the first center. Then d would be chosen as the second center. This solution has a cost of 2 for k-center which is even optimal for this instance. However, for any sum-based objective, the solution is by a factor of $\Omega(m)$ worse than the optimum solution: All points at c have to pay for being assigned to d, while in the optimum solution $\{a, c\}$, only two points pay (namely, b and d).

Furthermore, we can also see at this simple example that picking the first center *arbitrarily* is not a good idea, just by the same argument: If we pick arbitrarily, we might spend one center on either b or d, and then the final solution is off by a factor of $\Omega(m)$, no matter how we pick the second center.

D^2-sampling addresses these issues. It chooses the first point *uniformly at random* from P. In the above mini example, this would mean that with probability $(2m-2)/m$, we choose a good first center. The remaining centers are then also chosen randomly. For the ith center, a point from P is chosen with a probability that is proportional to *its current cost*. That has the following effect: Points that are currently very costly are more likely to be chosen. However, if there are *many* points at some area (for example, in the middle of an optimum cluster), then it becomes very likely that one of these points is chosen – even though there might be a single point somewhere which has a higher current cost.

D2-sampling($P \subset \mathbb{R}^d$, $k \in \mathbb{N}^{\geq 1}$)

1. Choose a point $x_1 \in P$ uniformly at random, set $C^1 = \{x_1\}$
2. **For** $i = 2$ to k **do**
3. Draw a point $x_i \in P$ according to the probability distribution

$$\frac{\min_{c \in C^{i-1}} ||x - c||^2}{\sum_{y \in P} \min_{c \in C^{i-1}} ||y - c||^2}$$

4. **Set** $C^i = C^{i-1} \cup \{x_i\}$
5. **Return** C^k

Again consider the mini example: Now if a is the first center, then (assuming unit distances between neighbored points) b costs 1, every point at c costs $2^2 = 4$, and d costs $3^2 = 9$. Points at a cost nothing. So the probability to choose a point at c in the second iteration is

$$\frac{4m}{1 + 4m + 9} = 1 - \frac{10}{4m},$$

which goes to 1 when m goes to infinity.

Now our mini example is simple. In general, k-means++ does not always find perfect centers or even a constant-factor approximation. From a theoretical point of view, it is worse than known approximation algorithms. But compared to Lloyd's algorithm, k-means++ does have a bounded approximation ratio, at least on expectation. Arthur and Vassilvitskii show the following:

Theorem 41. *Let C be the solution computed by* **D2-sampling** *on $P \subset \mathbb{R}^d$ and $k \in \mathbb{N}^{\geq 1}$. Note that C is a random variable. The expected cost of C is a $\mathcal{O}(\log k)$-approximation, more precisely, it holds*

$$E[\text{dist}^2(P, C)] \leq 8(\ln k + 2)\,\text{dist}^2(P, C^*)$$

where C^* is an optimal solution for the input P, k, and ln *is the natural logarithm.*

3.3.1 Bicriteria approximation algorithm

We first give a bicriteria result for D^2-sampling which is due to Aggarwal, Deshpande and Kannan [ADK09]. An (α, β)-bicriteria approximation algorithm for a standard clustering problem is an algorithm that is given P and k and computes a solution of cost $\alpha\,\text{dist}^2(P, C^*)$, where C^* is an optimal solution for the k-means problem on (P, k), but with $\beta \cdot k$ centers. Bicriteria algorithms are also used in other contexts; the defining property is that the solution space is extended by relaxing some condition in a controlled way and obtaining a solution of a cost that is good compared to an optimal solution without the relaxation.

For our setting, this means that D^2-sampling is used with a $t \geq k$ to obtain a solution that is good compared to the best solution with k centers. For the analysis, we

will compare the computed solution with an arbitrary fixed optimum solution $C^* = \{c_1^*, \ldots, c_k^*\}$. As we did in the proof for k-center with outliers, we again use $\mathcal{A}$ to denote a fixed clustering induced by this optimal solution.

For deriving the technical details of the analysis, we will have to compute probabilities of hitting a point from a specific subset of P multiple times. So before we start, it is good to observe that for any $S \subseteq P$ and any sample x drawn with D^2-sampling based on center set C^{i-1}, we have

$$\begin{aligned}\text{Prob}(x \in S) = \sum_{y \in S} \text{Prob}(x = y) &= \sum_{y \in S} \frac{\min_{c \in C^{i-1}} ||y - c||^2}{\sum_{z \in P} \min_{c \in C^{i-1}} ||z - c||^2} \\ &= \frac{\text{dist}^2(S, C^{i-1})}{\text{dist}^2(P, C^{i-1})}.\end{aligned}$$

The general idea of D^2-sampling is to try and find clusters that are still expensive and cover them by drawing a good center for them. The following definition states what we consider to be an expensive cluster for the course of the bicriteria analysis.

Definition 42. *For a center set $C \subset \mathbb{R}^d$, we say that a cluster $A_j \in \mathcal{A}$ is* good *if*

$$\text{dist}^2(A_j, C) \leq 10 \cdot \text{dist}^2(A_j, \mu(A_j)).$$

Otherwise, we say A_j is bad *with respect to C. Furthermore, at the beginning of iteration i of D^2-sampling, we define $GOOD_i$ as the set of clusters that are good with respect to C^{i-1}, and $BAD_i = \mathcal{A} \backslash GOOD_i$.*

So the probability to draw a point from a subset S is the share of the total cost that it is currently contributing.

Observe that C is a 10-approximation if all clusters are good. This is the aim of the algorithm: Continue to sample points until all clusters are good. However, that is not necessarily possible. Assume there is a cluster which has zero cost in the optimum, but right now costs a little but, but much less than the overall cost of the solution. Then D^2-sampling will most likely not find a point from this cluster because its probability mass is not very large. But then again, this does not really matter because the overall cost is not affected by this cluster, even though it is a bad cluster. So, even if some clusters are still bad, we might have a good approximation if these clusters do not contribute much to the overall cost. We are interested to analyze what happens if there are bad clusters and these cause a lot of cost. The next lemma says that if the solution is not a 20-approximation, then the point that we sample next is from a bad cluster with constant probability.

Lemma 43. *In iteration i of D^2-sampling, either* $\text{dist}^2(P, C^{i-1}) \leq 20 \cdot \text{dist}^2(P, C^*)$, *or the probability that x_i is from BAD_i is at least* $1/2$.

Proof. We show that if $\text{dist}^2(P, C^{i-1}) > 20 \cdot \text{dist}^2(P, C^*)$, then

$$\text{Prob}(x_i \in \bigcup_{A \in BAD_i} A) \geq \frac{1}{2}.$$

Observe that

$$\sum_{A \in GOOD_i} \text{dist}^2(A, C^{i-1}) \leq \sum_{A \in GOOD_i} 10 \cdot \text{dist}^2(A, \mu(A)) \leq 10 \cdot \text{dist}^2(P, C^*)$$

by the definition of good clusters and the fact that the good cluster are a subset of all clusters. So if $\text{dist}^2(P, C^{i-1}) > 20 \cdot \text{dist}^2(P, C^*)$, then

$$\sum_{A \in GOOD_i} \text{dist}^2(A, C^{i-1}) \leq 10 \cdot \text{dist}^2(P, C^*) = \frac{1}{2} \cdot 20 \cdot \text{dist}^2(P, C^*) \leq \frac{1}{2} \text{dist}^2(P, C^{i-1}).$$

This means that we can bound the probability to sample a point from a bad cluster from below by

$$\text{Prob}(x_i \in \bigcup_{A \in BAD_i} A) = \frac{\sum_{A \in BAD_i} \text{dist}^2(A, C^{i-1})}{\text{dist}^2(P, C^{i-1})} \geq \frac{1}{2}.$$

□

So with each sample, we have a constant probability to get a point from a cluster that is bad. However, we still have to figure out the probability that this sample is actually a good center for that bad cluster. In order to do that, we identify a subset of points of any cluster which contains only good centers. It is called $B(A_j, \alpha)$, and it is defined in the following definition.

Definition 44. *For any $A_j \in \mathcal{A}$, define*

$$(r(A_j))^2 := \frac{1}{|A_j|} \text{dist}^2(A_j, \mu(A_j)) = \frac{1}{|A_j|} \sum_{x \in A_j} ||x - \mu(A_j)||^2$$

and

$$B(A_j, \alpha) := \{x \in A_j \mid ||x - \mu(A_j)|| \leq \alpha \cdot r(A_j)\}.$$

So $r^2(A_j)$ is the average cost of a point in A_j, and $B(A_j, \alpha)$ contains all points whose cost is at most α^2 times the average cost. The following lemma gives the evidence that all points in $B(A_j, \alpha)$ are good centers for A_j (if α is a constant).

Lemma 45. *Let $A_j \in \mathcal{A}$ and $z \in B(A_j, \alpha)$. Then*

$$\text{dist}^2(A_j, z) \leq (\alpha^2 + 1) \text{dist}^2(A_j, \mu(A_j)).$$

Proof. The lemma is a consequence of Fact 39 together with the definitions of $B(A_j, \alpha)$ and $r(A_j)$:

$$\begin{aligned}
\text{dist}^2(A_j, z) &\overset{\text{Fact 39}}{=} \text{dist}^2(A_j, \mu(A_j)) + |A_j| \cdot ||\mu(A_j) - z||^2 \\
&\leq \text{dist}^2(A_j, \mu(A_j)) + |A_j| \cdot \alpha^2 \cdot (r(A_j))^2 \\
&= \text{dist}^2(A_j, \mu(A_j)) + |A_j| \cdot \alpha^2 \cdot \frac{\text{dist}^2(A_j, \mu(A_j))}{|A_j|} \\
&= (\alpha^2 + 1) \cdot \text{dist}^2(A_j, \mu(A_j)).
\end{aligned}$$

□

Now that we have seen that points in $B(A_j, \alpha)$ are good centers, we want to lower bound the probability of drawing one of them. First, we observe that there are many points in $B(A_j, \alpha)$. This is true because if there were many points outside of $B(A_j, \alpha)$, they together would cost too much; more than the cost of A_j, which is impossible.

Lemma 46.

$$|B(A_j), \alpha| \geq \left(1 - \frac{1}{\alpha^2}\right) \cdot |A_j|$$

Proof. Assume that

$$\begin{aligned} & |B(A_j), \alpha| \leq \left(1 - \frac{1}{\alpha^2}\right) \cdot |A_j| \\ \Leftrightarrow \quad & |A_j \backslash B(A_j), \alpha| \geq \frac{1}{\alpha^2} \cdot |A_j|, \end{aligned}$$

i.e., many points lie outside of $B(A_j, \alpha)$. By the definition of $B(A_j, \alpha)$, for all points $x \notin B(A_j, \alpha)$, $||x - \mu(A_j)||^2 > \alpha^2 (r(A_j))^2$. Thus, our assumption the cost of the points outside of $B(A_j, \alpha)$ alone already cost

$$\begin{aligned} \sum_{x \in A \backslash B(A_j, \alpha)} ||x - \mu(A_j)||^2 > & \frac{1}{\alpha^2} \cdot |A_j| \cdot \alpha^2 (r(A_j))^2 \\ = & \frac{1}{\alpha^2} \cdot |A_j| \cdot \alpha^2 \frac{\text{dist}^2(A_j, \mu(A_j))}{|A_j|} \\ = & \, \text{dist}^2(A_j, \mu(A_j)). \end{aligned}$$

Thus, under the assumption that many points lie outside of $B(A_j, \alpha)$, we have

$$\text{dist}^2(A_j, \mu(A_j)) \geq \sum_{x \in A \backslash B(A_j, \alpha)} ||x - \mu(A_j)||^2 > \text{dist}^2(A_j, \mu(A_j)),$$

which is a contradiction, and that proves the lemma. □

Second, we show that the points in $B(A_j, \alpha)$ are not only many, but also have a high total cost.

Lemma 47.

$$\text{Prob}(x_i \in B(A_j, \alpha) \mid x_i \in A_j \wedge A_j \in BAD_i) \geq \frac{1}{10} \cdot \left(1 - \frac{1}{\alpha^2}\right) \cdot (3 - \alpha)^2.$$

Proof. The probability that we want to compute is:

$$\text{Prob}(x_i \in B(A_j, \alpha) \mid x_i \in A_j \wedge A_j \in BAD_i) = \frac{\text{dist}^2(B(A_j, \alpha), C^{i-1})}{\text{dist}^2(A_j, C^{i-1})}.$$

We do that by using the following statements.

1. If we let $c_j = \arg\min_{c \in C^{i-1}} ||\mu(A_j) - c||^2$ be a center in C^{i-1} closest to $\mu(A_j)$, then $\text{dist}^2(A_j, C^{i-1}) \leq \text{dist}^2(A_j, c_j) = \text{dist}^2(A_j, \mu(A_j)) + |A_j| \cdot ||\mu(A_j) - c_j||^2 = |A_j| \cdot ((r(A_j))^2 + ||\mu(A_j) - c_j||^2)$ by Fact 39 and the definition of $r(A_j)$.

2. We observe that

$$\begin{aligned}|A_j| \cdot ||\mu(A_j) - c_j||^2 &= \text{dist}^2(A_j, c_j) - \text{dist}^2(A_j, \mu(A_j)) \\ &\geq \text{dist}^2(A_j, C^{i-1}) - \text{dist}^2(A_j, \mu(A_j)) \geq 9\,\text{dist}^2(A_j, \mu(A_j)),\end{aligned}$$

   which implies that $||\mu(A_j) - c_j||^2 \geq 9 \cdot \frac{\text{dist}^2(A_j, \mu(A_j))}{|A_j|} = 9(r(A_j))^2$.

3. If we pick any $c \in C^{i-1}$ and any $x \in B(A_j, \alpha)$, it holds that:

$$\begin{aligned}||x - c|| \geq ||c - \mu(A_j)|| - ||x - \mu(A_j)|| &\geq ||\mu(A_j) - c_j|| - ||x - \mu(A_j)|| \\ &\geq ||\mu(A_j) - c_j|| - \alpha \cdot r(A_j),\end{aligned}$$

   where the first inequality follows by the triangle inequality, the second follows by the definition of c_j, and the third follows since $x \in B(A_j, \alpha)$. This implies that

$$\text{dist}^2(B(A_j, \alpha), C^{i-1}) \geq |B(A_j, \alpha)| \cdot (||\mu(A_j) - c_j|| - \alpha \cdot r(A_j))^2.$$

4. $|B(A_j), \alpha| \geq \left(1 - \frac{1}{\alpha^2}\right) \cdot |A_j|$ holds by Lemma 46.

We use these facts to observe that

$$\begin{aligned}\frac{\text{dist}^2(B(A_j, \alpha), C^{i-1})}{\text{dist}^2(A_j, C^{i-1})} &\overset{1}{\geq} \frac{\text{dist}^2(B(A_j, \alpha), C^{i-1})}{|A_j| \cdot \left((r(A_j))^2 + ||\mu(A_j) - c_j||^2\right)} \\ &\overset{3}{\geq} \frac{|B(A_j, \alpha)| \cdot \left(||\mu(A_j) - c_j|| - \alpha \cdot r(A_j)\right)^2}{|A_j| \cdot \left((r(A_j))^2 + ||\mu(A_j) - c_j||^2\right)} \\ &\overset{4}{\geq} \frac{\left(1 - \frac{1}{\alpha^2}\right) \cdot |A_j| \cdot \left(||\mu(A_j) - c_j|| - \alpha \cdot r(A_j)\right)^2}{|A_j| \cdot \left((r(A_j))^2 + ||\mu(A_j) - c_j||^2\right)} \\ &= \frac{\left(1 - \frac{1}{\alpha^2}\right) \cdot \left(||\mu(A_j) - c_j|| - \alpha \cdot r(A_j)\right)^2}{(r(A_j))^2 + ||\mu(A_j) - c_j||^2}\end{aligned}$$

Now, although we do not know $||\mu(A_j) - c_j||$, we know that $||\mu(A_j) - c_j|| \geq 3r(A_j)$ holds by 2. If we consider $\frac{\cdot(||\mu(A_j) - c_j|| - \alpha \cdot r(A_j))^2}{(r(A_j))^2 + ||\mu(A_j) - c_j||^2}$ as a function in $||\mu(A_j) - c_j||$, we observe (e.g., by computing its first and second derivative), that it is monotonically increasing for $||\mu(A_j) - c_j|| \geq r(A_j)$. Thus, we can replace $||\mu(A_j) - c_j||$ by $3r(A_j)$ to obtain the lower bound

$$\frac{\left(1 - \frac{1}{\alpha^2}\right) \cdot (3r(A_j) - \alpha \cdot r(A_j))^2}{10(r(A_j))^2} = \frac{1}{10} \cdot \left(1 - \frac{1}{\alpha^2}\right) \cdot (3 - \alpha)^2.$$

□

After this fairly technical lemma, we have all ingredients to show that with constant probability, a new sample converts at least one bad cluster into a good cluster (as long as the solution is not good already).

Corollary 48. *Assume that in iteration i it still holds that* $\text{dist}^2(P, C^{i-1}) \geq 20\,\text{dist}^2(P, C^*)$. *Then the probability that x_i is in $B(A_j, 1.44225)$ for an $A_j \in BAD_i$ is at least* 0.126.

Proof. With probability $1/2$, x_i is from a bad cluster which we name A_j, and with probability at least $\frac{1}{10} \cdot \left(1 - \frac{1}{\alpha^2}\right) \cdot (3-\alpha)^2$, it is in $B(A_j, \alpha)$. Now for $\alpha = 1.4425$, we get

$$\frac{1}{2} \cdot \frac{1}{10} \cdot \left(1 - \frac{1}{1.4425^2}\right) \cdot (3 - 1.4425)^2 \geq 0.063.$$

□

Corollary 48 implies that the expected number of samples that we need to convert at least one bad cluster into a good cluster is $\frac{1}{0.063} \leq 16$. By linearity of expectation and Markov's inequality, we conclude that the probability that after $32k$ samples there are still bad clusters *and* the cost is more than $20 \cdot \text{dist}^2(P, C^*)$ is at most $1/2$. Details on this are discussed in the tutorial about Problem Set 07.

Theorem 49. *For $t = 32k$, `D2-sampling`(P,t) computes a 20-approximation for the k-means problem with probability at least $1/2$.*

3.3.2 A glimpse on the analysis of k-means++

We now consider to lemmata that motivate why D^2-sampling proceeds in the way that it does. We will skip the main proof, though. The following lemma is an immediate consequence of Fact 39. It shows that the first step of `D2-sampling` is a good idea.

Lemma 50. *Let $S \subset \mathbb{R}^d$ be a point set and let $x \in S$ be a point chosen uniformly at random from S. Then*

$$E[\text{dist}^2(S, x)] = 2 \cdot \text{dist}^2(S, \mu(S)).$$

Proof.

$$\begin{aligned}
E[\text{dist}^2(S,x)] &= \sum_{y \in S} \frac{1}{|S|} \text{dist}^2(S, y) \\
&= \frac{1}{|S|} \sum_{y \in S} \sum_{z \in S} ||y - z||^2 \\
&\overset{\text{Fact 39}}{=} \frac{1}{|S|} \sum_{y \in S} \left[\sum_{z \in S} ||y - \mu(S)||^2 + |P| \cdot ||z - \mu(S)||^2 \right] \\
&= \sum_{y \in S} ||y - \mu(S)||^2 + \sum_{z \in S} ||z - \mu(S)||^2 \\
&= 2\,\text{dist}^2(S, \mu(S)).
\end{aligned}$$

□

Now for the remaining steps, the goal of D^2-sampling is to hit clusters that are not 'covered' yet, i.e., from which we have not chosen a point as a center yet. Now even if this succeeds (it does not succeed necessarily, which causes the super-constant approximation ratio), we still have to show that the point that we get from such an uncovered cluster is a good center for that cluster. The following lemma provides that evidence. The guarantee is weaker than in Lemma 50. That makes sense: The probability distribution is somewhat skewed, so that we are more likely to draw points that are further away from the mean of the cluster. However, as the following lemma states, this effect is small enough and we still get a constant-factor approximation for S on expectation.

Lemma 51. *Let $S \subset \mathbb{R}^d$ be a point set, let $C \subset \mathbb{R}^d$ be an arbitrary finite set and let $x \in S$ be a point chosen randomly from S according to the probability distribution where each point $x \in S$ has probability*

$$\frac{\min_{c \in C} ||x - c||^2}{\sum_{y \in S} \min_{c \in C} ||y - c||^2}.$$

Then it holds that

$$E[\text{dist}^2(S, C \cup \{x\})] \leq 8 \, \text{dist}^2(S, \mu(S)).$$

Proof. We approach the proof in the same way as in Lemma 50, but now have non-uniform probabilities for the sampling, and we consider $C \cup \{x\}$ (instead of only x). If $x \in S$ is the sampled point, then the cost after adding it to C is

$$\sum_{z \in S} \min_{c \in (C \cup \{x\})} ||z - c||^2.$$

Thus the expression that we get for our expected cost is

$$E[\text{dist}^2(S, C \cup \{x\})] = \sum_{y \in S} \frac{\min_{c \in C} ||y - c||^2}{\sum_{z \in S} \min_{c \in C} ||z - c||^2} \sum_{z \in S} \min_{c \in (C \cup \{y\})} ||z - c||^2. \quad (3.1)$$

Before computing it, we find an upper bound for the term $\min_{c \in C} ||y - c||^2$. The trick is the following. For all $z \in S$, it holds by the relaxed triangle inequality (Lemma 38) that

$$\min_{c \in C} ||y - c||^2 \leq ||y - c(z)||^2 \leq 2||y - z||^2 + 2 \min_{c \in C} ||z - c||^2,$$

where $c(z)$ is a center closest to z in C. Now we sum this inequality up over all $z \in S$ and divide the resulting inequality by $|S|$ to obtain that

$$\min_{c \in C} ||y - c||^2 \leq \frac{2}{|S|} \sum_{z \in S} (||y - z||^2 + \min_{c \in C} ||z - c||^2).$$

We use this to bound the term in Equation 3.1 and get

$$\sum_{y \in S} \frac{\frac{2}{|S|} \sum_{z \in S} (||y - z||^2 + \min_{c \in C} ||z - c||^2)}{\sum_{z \in S} \min_{c \in C} ||z - c||^2} \sum_{z \in S} \min_{c \in (C \cup \{x\})} ||z - c||^2.$$

Now we split this up into two big sums and bound each individually. The first one is

$$\frac{2}{|S|}\sum_{y\in S}\frac{\sum_{z\in S}||y-z||^2}{\sum_{z\in S}\min_{c\in C}||z-c||^2}\sum_{z\in S}\min_{c\in(C\cup\{y\})}||z-c||^2, \tag{3.2}$$

and the second one is

$$\frac{2}{|S|}\sum_{y\in S}\frac{\sum_{z\in S}\min_{c\in C}||z-c||^2}{\sum_{z\in S}\min_{c\in C}||z-c||^2}\sum_{z\in S}\min_{c\in(C\cup\{x\})}||z-c||^2. \tag{3.3}$$

For (3.2), we use that $\min_{c\in(C\cup\{y\})}||z-c||^2 \leq \min_{c\in C}||z-c||^2$ to get that

$$\begin{aligned}
& \frac{2}{|S|}\sum_{y\in S}\frac{\sum_{z\in S}||y-z||^2}{\sum_{z\in S}\min_{c\in C}||z-c||^2}\sum_{z\in S}\min_{c\in(C\cup\{y\})}||z-c||^2 \\
\leq\ & \frac{2}{|S|}\sum_{y\in S}\sum_{z\in S}||y-z||^2 \\
\overset{\text{Fact 39}}{=}\ & \frac{2}{|S|}\left(\sum_{y\in S}\operatorname{dist}^2(S,\mu(S))+\sum_{y\in S}||S|\cdot||\mu(S)-y||^2\right) \\
=\ & 4\operatorname{dist}^2(S,\mu(S)).
\end{aligned}$$

For (3.3), we get via $\min_{c\in(C\cup\{y\})}||z-c||^2 \leq ||z-y||^2$ that

$$\begin{aligned}
& \frac{2}{|S|}\sum_{y\in S}\sum_{z\in S}\min_{c\in(C\cup\{x\})}||z-c||^2 \\
\leq\ & \frac{2}{|S|}\sum_{y\in S}\sum_{z\in S}||z-y||^2 \\
\overset{\text{Fact 39}}{=}\ & \frac{2}{|S|}\left(\sum_{y\in S}||z-\mu(S)||^2+|S|\cdot||z-\mu(S)||^2\right) \\
=\ & 4\operatorname{dist}^2(S,\mu(S)).
\end{aligned}$$

Adding both terms proves the lemma. □

Now based on Lemma 50 and Lemma 51, the proof of the approximation guarantee for k-means++ proceeds by showing that the expectation of the current cost develops favorably during the k iterations. Morally, what happens is that if a new cluster is hit by the sampling, then this new cluster is constant-approximated, and this happens with a good probability. However, the probability to hit a already covered cluster again is large enough to raise the expecation from constant to H_k, which translates into the $\Theta(\log k)$ approximation guarantee. We skip the details of this rather involved inductive proof.

3.4 Dimensionality Reduction for k-means

There are two very popular methods for reducing the dimension of points in $\mathbb{R}^d$: Random projections / JL projections and principal component analysis / SVD. In this

section, we discuss two important results about these two methods with respect to the k-means cost function.

Before we do that, we discuss how clustering problems are usually modeled as matrix problems. This is very helpful for describing linear projections. We start by writing the points of the input point set into the rows of a matrix A, i.e., for a point set $P = \{x_1, \ldots, x_n\}$, we construct a matrix $A \in \mathbb{R}^{n \times d}$ by

$$A = \begin{pmatrix} — & x_1 & — \\ — & x_2 & — \\ & \cdots & \\ — & x_n & — \end{pmatrix}.$$

Then we encode any possible solution $C \subset \mathbb{R}^d$, $|C| = k$ as a matrix as well, by writing the closest center to point x_i in row i of a matrix $S(C)$:

$$S(C) = \begin{pmatrix} — & \arg\min_{c \in C} ||x_1 - c|| & — \\ — & \arg\min_{c \in C} ||x_2 - c|| & — \\ & \cdots & \\ — & \arg\min_{c \in C} ||x_n - c|| & — \end{pmatrix}.$$

So any solution C translates to a matrix S of rank k, which in fact consists of copies of the same k points. Now we can express $\text{dist}^2(P, C)$ in terms of A and $S(C)$:

$$\text{dist}^2(A, C) = \sum_{i=1}^{n} \min_{c \in C} ||x_i - c||^2 = \sum_{i=1}^{n} (x_i - \arg\min_{c \in C} ||x_1 - c||)^2 = ||A - S(C)||_F^2,$$

where $|| \cdot ||_F$ is the Frobenius norm (see Definition 4).

A linear map now is a matrix $X \in \mathbb{R}^{d \times d'}$, where d' is the dimension of the target space. It may be equal to d, but that does not have to be the case.

3.4.1 The Johnson-Lindenstrauss Lemma

The first one is based on the famous *Johnson-Lindenstrauss Lemma*. The following wording is inspired by Dasgupta and Gupta [DG03], who give a notably concise proof of the Johnson-Lindenstrauss Lemma that is based on elementary building blocks. We still skip the proof and focus on the application of the JL Lemma to the k-means problem.

Theorem 52 (Johnson-Lindenstrauss Lemma [JL84])**.** *For any $P \subset \mathbb{R}^d$ with $|P| = n$, any constant $\varepsilon \in (0, 1)$, and any number $d' \geq d_0(n, \varepsilon)$ for some specific $d_0(n, \varepsilon) \in \Theta(\varepsilon^{-2} \log n)$, there exists a linear map $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ that satisfies for all $x, y \in P$ that*

$$(1 - \varepsilon)||x - y||^2 < ||f(x) - f(y)||^2 < (1 + \varepsilon)||x - y||^2.$$

Furthermore, such a map can be found by a randomized algorithm with polynomial running time and constant success probability.

A linear map satisfying Theorem 52 can be constructed in different (usually probabilistic) ways, and finding maps with good additional properties is the topic of ongoing research. The probabilistic constructions have in common that one does not need to know anything about the points except their number, which is a very strong advantage of JL projections. We cite one concrete construction that is due to Achlioptas [Ach03] and gives a random projection matrix that only consists of 1s and (-1)s.

Theorem 53. *Let $P \subset \mathbb{R}^d$ be a point set represented by a matrix $A \in \mathbb{R}^{n\times d}$. Given $\varepsilon, \beta > 0$, set*

$$d_0 := \frac{4+2\beta}{\varepsilon^2/2 - \varepsilon^3/3} \log n.$$

For any integer $d' \geq d_0$, let $R \in \mathbb{R}^{d\times d'}$ be a random matrix, where each entry r_{ij} is drawn from the following probability distribution:

$$r_{ij} = \begin{cases} +1 & \text{with probability } 1/2 \\ -1 & \text{with probability } 1/2. \end{cases}$$

Let $E = \frac{1}{\sqrt{d}} AR$ and let $f : \mathbb{R}^d \to \mathbb{R}^k$ map the ith row of A to the ith row of E. Then, with probability at least $1 - n^{-\beta}$, for all $x, y \in P$:

$$(1-\varepsilon)||x-y||^2 \leq ||f(x) - f(y)||^2 \leq (1+\varepsilon)||x-y||^2.$$

If one wants to obtain a result exactly like Theorem 52, i.e., one wishes to approximate the distances between *all* pairs of points up to a factor of $\pm\varepsilon$, then the target dimension $\Theta(\varepsilon^{-2} \log n)$ in the Johnson Lindenstrauss lemma is proven to be nearly tight.

How can we use Theorem 52 in the context of the k-means problem? We do this by using an alternative characterization of the k-means cost function. The following lemma states that we can express the optimal 1-means cost of a point set as a function of the pairwise distances of the points in the set.

Lemma 54. *Let $P \subset \mathbb{R}^d$ be a finite point set with centroid $\mu(P)$. It holds that*

$$\operatorname{dist}^2(P, \mu(P)) = \frac{1}{2|P|} \sum_{x\in P} \sum_{y \in P} ||x-y||^2.$$

Proof. We show the inequality backwards. For that, we the inner sum $\sum_{y\in P} ||x-y||^2$ as the clustering of P with center x and can then apply Lemma 39. This yields that

$$\begin{aligned} \frac{1}{2|P|} \sum_{y\in P} \sum_{x\in P} ||x-y||^2 &= \frac{1}{2|P|} \sum_{x\in P} \left[\left(\sum_{y\in P} ||y-\mu(P)||^2 \right) + |P| \cdot ||x-\mu||^2 \right] \\ &= \frac{1}{2} \sum_{y\in P} ||y-\mu(P)||^2 + \frac{1}{2|P|} \sum_{x\in P} |P| \cdot ||x-\mu||^2 \\ &= \sum_{y\in P} ||y-\mu||^2 = \operatorname{dist}^2(P, \mu(P)), \end{aligned}$$

and that is what we wanted to show. □

Now we want to relate Lemma 54 to the k-means cost function. Notice that we can do that as long as the centers are centroids of the induced clustering. To account for this detail, we define the following variation of the k-means problem.

Definition 55. *Given a point set P and a number $k \in \mathbb{N}^{\geq 1}$, we define the* partition-based k-means problem *as the task to find a partitioning $P_1, \dots, P_k$ of P (i.e., $\cup_{i=1}^{k} P_i = P$ and $P_i \cap P_j = \emptyset$ for all $i, j \in [k]$ with $i \neq j$) that minimizes*

$$\operatorname{dist}^2((P_i)_{i=1}^k) := \sum_{i=1}^{k} \operatorname{dist}^2(P_i, \mu(P_i)).$$

The partition-based k-means problem has a restricted set of possible solutions compared to the general k-means problem. For solving the problem optimally, this distinction is irrelevant: Because of Lemma 39, the centers of any optimal solution are the centroids of the clustering that they induce. We differentiate the two problems because in the following lemma, we want to claim that projecting points with a JL projection approximately preserves the cost of *any possible solution*. This is true for the partition-based k-means problem, but not necessarily for the k-means problem.

Lemma 56. *Let $P \subset \mathbb{R}^d$ be a point set with n points and let $\varepsilon \in (0, 1)$ and $d' \geq d_0(n, \varepsilon)$ for the $d_0 \in \Theta(\varepsilon^{-2} \log n)$ arising from Theorem 52. There exists a linear map $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ such that for all possible partitionings $P_1, \dots, P_k$ of P,*

$$|\operatorname{dist}^2((P_i)_{i=1}^k) - \operatorname{dist}^2((f(P_i))_{i=1}^k)| \leq \varepsilon \cdot \operatorname{dist}^2((P_i)_{i=1}^k),$$

where $f(P_i) := \{f(x) \mid x \in P\}$.

Proof. We apply Theorem 52 with precision parameter ε, gaining that there exists a map $f : \mathbb{R}^d \to \mathbb{R}^{d_0(n,\varepsilon)}$ such that $(1-\varepsilon)||x-y||^2 < ||f(x)-f(y)||^2 < (1+\varepsilon)||x-y||^2$ holds for all $x, y \in P$, or, equivalently, it holds for all $x, y \in P$ that

$$\left| ||x-y||^2 - ||f(x)-f(y)||^2 \right| \leq \varepsilon \cdot ||x-y||^2. \tag{3.4}$$

Now we observe that

$$\operatorname{dist}^2((P_i)_{i=1}^k) = \sum_{i=1}^{k} \sum_{x \in P_i} ||x - \mu(P_i)||^2 \overset{\text{Lemma 54}}{=} \sum_{i=1}^{k} \frac{1}{2|P_i|} \sum_{x \in P_i} \sum_{y \in P_i} ||x-y||^2,$$

and similarly,

$$\begin{aligned} \operatorname{dist}^2((f(P_i))_{i=1}^k) &= \sum_{i=1}^{k} \frac{1}{2|f(P_i)|} \sum_{a \in f(P_i)} \sum_{b \in f(P_i)} ||x-y||^2 \\ &= \sum_{i=1}^{k} \frac{1}{2|P_i|} \sum_{x \in P_i} \sum_{y \in P_i} ||f(x)-f(y)||^2. \end{aligned}$$

Thus, the difference between the two cost functions can be bounded in the following way:

$$|\operatorname{dist}^2((P_i)_{i=1}^k) - \operatorname{dist}^2((f(P_i))_{i=1}^k)|$$

$$
\begin{aligned}
&= \left| \sum_{i=1}^{k} \frac{1}{2|P_i|} \sum_{x \in P_i} \sum_{y \in P_i} ||x-y||^2 - \sum_{i=1}^{k} \frac{1}{2|P_i|} \sum_{x \in P_i} \sum_{y \in P_i} ||f(x)-f(y)||^2 \right| \\
&= \left| \sum_{i=1}^{k} \frac{1}{2|P_i|} \sum_{x \in P_i} \sum_{y \in P_i} \left(||x-y||^2 - ||f(x)-f(y)||^2 \right) \right| \\
&\leq \sum_{i=1}^{k} \frac{1}{2|P_i|} \sum_{x \in P_i} \sum_{y \in P_i} \left| ||x-y||^2 - ||f(x)-f(y)||^2 \right| \\
&\leq \sum_{i=1}^{k} \frac{1}{2|P_i|} \sum_{x \in P_i} \sum_{y \in P_i} \varepsilon \cdot ||x-y||^2 = \varepsilon \cdot \mathrm{dist}^2((P_i)_{i=1}^k),
\end{aligned}
$$

where the last inequality follows from Inequality (3.4) above. □

Lemma 56 says that we can project any point set P with n points to a $d_0(n, \varepsilon)$-dimensional space without losing much in the k-means cost function. This can be used for many different purposes; the first one is to reduce the running time of an algorithm for the k-means problem.

`JL-dimred-k-means`$(P \subset \mathbb{R}^d,\ k \in \mathbb{N}^{\geq 1},\ \varepsilon \in (0, \frac{1}{2}))$

1. Compute $\varepsilon' = \varepsilon/6\alpha$ and $d' := d_0(n, \varepsilon')$ (according to Theorem 52)
2. Compute $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ and $f(P) \subset \mathbb{R}^{d'}$ (according to Theorem 52)
3. Use an α-approximation algorithm for the partition-based k-means problem on $f(P)$ to obtain a partitioning $(f_i)_{i=1}^k$ of $f(P)$
4. Translate $(f_i)_{i=1}^k$ into the partitioning $(P_i)_{i=1}^k$ where $P_i = \{x \mid f(x) \in f_i\}\ \forall i \in [k]$
5. **Return** $(P_i)_{i=1}^k$

Corollary 57. *Algorithm* `JL-dimred-k-means` *computes an $(\alpha + \varepsilon)$-approximation for the partition-based k-means problem with constant probability.*

Proof. We assume that the randomized algorithm was successful in producing a mapping with the desired properties. Let $P_1^*, \ldots, P_k^*$ be an optimal solution to the partition-based k-means problem; this happens with constant probability. By Lemma 56,

$$\mathrm{dist}^2((f(P_i)^*)_{i=1}^k) \leq (1+\varepsilon') \cdot \mathrm{dist}^2((P_i^*)_{i=1}^k).$$

Thus, the optimal cost of any solution to the partition-based k-means problem on $f(P)$ costs at most $(1+\varepsilon') \cdot \mathrm{dist}^2(((P_i)^*)_{i=1}^k)$. Let $f_1, \ldots, f_k$ be the α-approximate solution on $f(P)$ that is computed. Since it is an α-approximation, its cost is at most $(1+\varepsilon') \cdot \alpha \cdot \mathrm{dist}^2(((P_i)^*)_{i=1}^k)$. Now we translate $(f_i)_{i=1}^k$ back and obtain the partitioning $(P_i)_{i=1}^k$ of P. Again by Lemma 56, we know that

$$
\begin{aligned}
&\mathrm{dist}^2(((P_i))_{i=1}^k) \geq (1-\varepsilon') \cdot \mathrm{dist}^2(((f_i))_{i=1}^k) \\
\Leftrightarrow\ &\mathrm{dist}^2(((P_i))_{i=1}^k) \leq \frac{1}{1-\varepsilon'} \cdot \mathrm{dist}^2(((f_i))_{i=1}^k) \leq (1+2\varepsilon') \cdot \mathrm{dist}^2(((f_i))_{i=1}^k)
\end{aligned}
$$

where the last inequality follows from $\varepsilon \leq \frac{1}{2}$. So we get that $\text{dist}^2(((P_i))_{i=1}^k) \leq (1 + 2\varepsilon') \cdot (1 + \varepsilon') \cdot \alpha \cdot \text{dist}^2(((P_i)^*)_{i=1}^k) \leq (1 + 6\varepsilon') \cdot \alpha \cdot \text{dist}^2(((P_i)^*)_{i=1}^k)$. Now the corollary follows since $(1 + 6\varepsilon')\alpha = \alpha + 6\varepsilon'\alpha = \alpha + \varepsilon$ by our choice of ε'. □

3.4.2 The Singular Value Decomposition

The singular value decomposition (SVD) is a matrix decomposition, i.e., a way to write A as a multiplication of matrices with beneficial properties. In contrast to other decompositions, the SVD exists for any nonzero matrix A. We will not show this, but concentrate on exploring the geometric properties of the SVD and its usefulness for reducing the dimension of k-means inputs. For this, we are actually less interested in matrix form of the decomposition, and more in the statement as given in the following theorem.

Theorem 58 ([Wat10], Theorem 5.8.11 on page 391, and [KV09], Theorem 1.3 on pages 163-165)**.** *Let $A \in \mathbb{R}^{n \times d}$ be a nonzero matrix with rank r. Then there exists an orthonormal basis $v_1, \ldots, v_d$ of $\mathbb{R}^d$ and an orthonormal basis $u_1, \ldots, u_n$ of $\mathbb{R}^n$ and positive values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$ such that*

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T.$$

Furthermore, it holds that

$$Av_i = \begin{cases} \sigma_i u_i & i = 1, \ldots, r \\ 0 & i = r+1, \ldots, d \end{cases} \quad \text{and} \quad u_i^T A = \begin{cases} \sigma_i v_i^T & i = 1, \ldots, r \\ 0 & i = r+1, \ldots, n \end{cases}.$$

The vectors u_i in Theorem 58 are called *left singular vectors*, the vectors v_i are called *right singular vectors* and the values σ_i are called *singular values*, which explains the term *singular value decomposition*. The actual matrix decomposition arises from writing the singular values and vectors into matrices: a diagonal matrix $D \in \mathbb{R}^{n \times d}$, and two orthogonal matrices $U \in \mathbb{R}^{n \times n}$, $V \in \mathbb{R}^{d \times d}$. This is how the matrices are constructed:

$$U = \begin{pmatrix} | & & | \\ u_1 & \cdots & u_n \\ | & & | \end{pmatrix}, \; D = \left(\begin{array}{ccc|c} \sigma_1 & \cdots & 0 & \\ \vdots & \ddots & \vdots & 0 \\ 0 & \cdots & \sigma_{\min\{n,d\}} & \\ \hline & 0 & & 0 \end{array}\right), \; V^T = \begin{pmatrix} — & v_1 & — \\ & \vdots & \\ — & v_d & — \end{pmatrix}$$

One can now observe that for U, D and V defined in this form, it holds that

$$\sum_{i=1}^{r} \sigma_i u_i v_i^T = UDV^T,$$

and thus, by Theorem 58, $A = UDV^T$.

3.4.3 Geometric interpretation

The aim of this paragraph is to interpret the matrix

$$A^{(k)} := \sum_{i=1}^{k} \sigma_i u_i v_i^T$$

for an $k \leq r$ which arises from only adding the first k summands of the SVD-based representation of A. We will discuss this without proofs and rather recall and interpret known facts from linear algebra.

First we look at the term $\sigma_i u_i$. From Theorem 58 we know that $\sigma_i u_i = Av_i$. What is Av_i? For a matrix that arises from writing the points in the rows Av_i is a vector that has the entries $x_j^T v_i$ for $j \in [n]$. Recall that v_i is a unit vector. Then the scalar product $x_j^T v_i$ is the length of the projection of x_j onto the 1-dimensional subspace spanned by v_i. So we get a vector which has the lengths of projecting all points onto v_i as its entries. Now in the sum, this vector is multiplied by v_i^T by a tensor product. This means that we get a matrix, and the rows of this matrix are the actual vectors arising from projecting the x_j onto v_i.

So each term $\sigma_i u_i v_i^T$ results in a matrix which contains the vectors resulting from projecting all rows of A, i.e., all points in P, onto the direction v_i. Recall that $v_1, \ldots, v_d$ form an basis of $\mathbb{R}^d$. Then the entries of $\sigma_i u_i$ are just the ith coordinate of all points in this basis. It thus makes sense that adding up all of them gives A. Leaving some of the summands out means that we ignore the contribution of those basis vectors to the points.

In comparison to an arbitrary basis, the basis $v_1, \ldots, v_d$ has a special property, and that will explain why $A^{(k)}$ is an interesting matrix. Recall that the singular values are ordered such that $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$. We observe that v_1, the right singular vector belonging to σ_1, maximizes the expression $||Av||$ among all unit vectors $v \in \mathbb{R}^d$. To see this, let v be an arbitrary unit vector. We can express any unit vector v by $v = \sum_{i=1}^{d} \alpha_i v_i$ with $\sum_{i=1}^{d} \alpha_i^2 = 1$.[1] Now we observe that

$$||Ax||^2 = \left|\left|\sum_{i=1}^{d} \alpha_i A v_i\right|\right|^2 = \left|\left|\sum_{i=1}^{d} \alpha_i \sigma_i u_i\right|\right|^2 = \sum_{i=1}^{d} \alpha_i^2 \sigma_i^2 ||u_i||^2 = \sum_{i=1}^{d} \alpha_i^2 \sigma_i^2.$$

This sum is maximized if $\alpha_1^2 = 1$ and $\alpha_i^2 = 0$ for $i \neq 1$. Thus, v_1 maximizes $||Ax||^2$ among all unit vectors, and thus it also maximizes $||Ax||$ among all unit vectors.

We have argued above that Av gives a vector with the projection lengths into direction v. Thus, $||Av||^2$ for any unit vector $v \in \mathbb{R}^d$ is the sum of squared projection lengths into direction v, and this expression is maximized by v_1.

Now notice that if a vector x is orthogonal to v_1, then its α_1 has to be zero. Thus, among all vectors that are orthogonal to v_1, v_2 maximizes $||Ax||^2$ because $\alpha_2 = 1$ and $\alpha_i = 0$ for $i \neq 2$ is the maximal choice. This argument works inductively.

[1] The sum of the squared coordinates is 1 because x is a unit vector and $1 = ||x||^2 = \left|\left|\sum_{i=1}^{d} \alpha_i v_i\right|\right|^2 = \sum_{i=1}^{d} \alpha_i^2 ||v_i||^2 = \sum_{i=1}^{d} \alpha_i^2$.

Fact 59. *It holds that*

$$\begin{aligned} v_1 &= \arg \max_{v \in \mathbb{R}^d, ||v||=1} ||Av||, \\ v_2 &= \arg \max_{v \in \mathbb{R}^d, ||v||=1, \langle v, v_1 \rangle = 0} ||Av||, \\ &\vdots \\ v_r &= \arg \max_{v \in \mathbb{R}^d, ||v||=1, \langle v, v_i \rangle = 0 \ \forall i=1,\ldots,r-1} ||Av||. \end{aligned}$$

for right singular vectors $v_1, \ldots, v_r \in \mathbb{R}^d$ as given by Theorem 58.

It is even true that a subspace which is spanned by $v_1, \ldots, v_k$ maximizes the sum of the squared projection lengths of the points in A among all k-dimensional subspaces. And this is why $A^{(k)}$ is interesting: It is a projection of the points in A to a k-dimensional subspace which preserves the largest part of the lengths among all projections of A to k-dimensional subspaces. We get a k-dimensional version of the points which is as close to A as possible. Indeed, $A^{(k)}$ is also the best approximation of A with respect to the Frobenius norm among all matrices of rank k, i.e.:

Fact 60.

$$||A - A^{(k)}||_F = \arg \min_{B \in \mathbb{R}^{n \times d}, \ \mathrm{rank}(B)=k} ||A - B||_F.$$

As a final note, we state that the subspace $V^{(k)}$ spanned by $v_1, \ldots, v_k$ is also a *best fit subspace* for A in the sense that it minimizes the sum of the squared distances of the points to the subspace. This follows from the Pythagorean Theorem (Fact 5) and the fact that $V^{(k)}$ maximizes the sum of the squared projection lengths.

Theorem 61 (KV09, Theorem 1.3 on pages 163+164)**.** *Find right singular vectors $v_i \in \mathbb{R}^d$ as in Theorem 59. For $k \leq d$, define $V_k := \mathrm{span}\{v_1, \ldots, v_k\}$ as the span of the first k of these vectors. Then, V_k is a* best fit subspace *of dimension k, i.e.,*

$$V_k = \arg \min_{V \subset \mathbb{R}^d, 0 \in V, \ \dim(V)=k} \sum_{i=1}^{n} \mathrm{dist}^2(x_i, V).$$

3.4.4 Using the SVD to compute a 2-approximation

The best fit subspace from Theorem 61 gives us a computable lower bound on the k-means cost of a point set because the squared distances to k centers cannot be larger than the squared distances to a subspace of dimension k.

Lemma 62. *Let $P := \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ be a set of points, and let V_k be a best fit subspace for P (see Theorem 61). Then*

$$\sum_{i=1}^{n} dist^2(x_i, V_k) = \mathrm{dist}^2(P, V_k) \leq \mathrm{dist}^2(P, C^*),$$

where C^* is an optimal solution for the k-means problem on P.

Proof. See tutorial (problem set 9, task 2). □

Drineas, Frieze, Kannan, Vempala and Vinay [DFK+04] use Lemma 62 to speed up algorithms for the k-means problem by using dimensionality reduction. Their key observation is that the k-means cost can be decomposed by the Pythagorean theorem, and then the two terms can be bound by Lemma 62 and the following lemma.

Lemma 63. *Let V be any linear subspace of $\mathbb{R}^d$ and for every point $x \in \mathbb{R}^d$ let $V(x) = \arg\min_{y \in V} d(x, y)$ be the point in V closest to x. Then for any set of points $P = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}^d$ the best k-means solution on P is at least as expensive as the best k-means solution on $\{V(x_1), \ldots, V(x_n)\}$ in V.*

Proof. See tutorial (problem set 9, task 3). □

Lemma 63 is even a bit more general than the statement that we need: It says that we can project a point set P and a center set C to any subspace V, and this will only decrease the distances between the P and C. In the following algorithm, the projection is to a specific subspace, namely V_k. Projecting to V_k means that we replace A by $A^{(k)}$ which we discussed in Section 3.4.3: it is the orthogonal projection of A to V_k.

`SVD-constant-dim-red`$(P \subset \mathbb{R}^d,\ k \in \mathbb{N}^{\geq 1})$

1. Store P in matrix A
2. Compute the SVD of A and compute $A^{(k)}$, interpret this as point set $P^{(k)} \subset \mathbb{R}^d$
3. Apply an α-approximation for the k-means problem to ($P^{(k)}$,k) to get a center set $C \subset \mathbb{R}^d$
4. **Return** C

Notice that the point set $P^{(k)}$ is still a set of points in $\mathbb{R}^d$. However, its intrinsic dimension is bounded by k. In order to profit from the lower intrinsic dimension, it might be necessary to actually compute a k-dimensional representation of $P^{(k)}$, depending on the algorithm that is used in line 3., and then translate the result back to $\mathbb{R}^d$.

Algorithm `SVD-constant-dim-red` can be combined with an optimal algorithm (i.e., $\alpha = 1$), and the resulting algorithm will have a running time that depends polynomially on d since d only influences the running time of the SVD computation. (And computing the SVD is possible in polynomial time.) It can also be combined with any approximation algorithm.

Theorem 64. *The center set C computed by algorithm `SVD-constant-dim-red` is a $(\alpha + 1)$-approximate solution for the k-means problem on (P,k).*

Proof. By the Pythagorean theorem, we can decompose the cost of P with the solution C computed by the algorithm as:

$$\text{dist}^2(P, C) = \text{dist}^2(P, V_k) + \text{dist}^2(P^{(k)}, C),$$

where we recall that $P^{(k)}$ is the projection of P to V_k computed in the 2. step of Algorithm `SVD-constant-dim-red` (notice that $C \subset V_k$ and that the vector between a point and its projection is orthogonal to any vector in V_k, i.e., also to the vector from the point's projection to its closest center in C).

The first term is bounded by Lemma 62. For the second term, let C^* be an optimal solution for the k-means problem on P, k, and let C_k^* be an optimal solution for the k-means problem on $P^{(k)}$, k. By Lemma 63, $\operatorname{dist}^2(P^{(k)}, C_k^*) \leq \operatorname{dist}^2(P, C^*)$. Since C is an α-approximation, we have $\operatorname{dist}^2(P^{(k)}, C) \leq \alpha \operatorname{dist}^2(P, C^*)$, which overall leads to $\operatorname{dist}^2(P, C) = \operatorname{dist}^2(P, V_k) + \operatorname{dist}^2(P^{(k)}, C) \leq (\alpha + 1) \cdot \operatorname{dist}^2(P, C^*)$. That completes the proof. □

Theorem 64 gives a neat way to approximately solve the k-means problem. Yet it does not have the very strong type of guarantee that we would ideally like to obtain for data reduction types: That for *every solution*, we get an approximation of the solution's cost. One can construct examples where it becomes clear that this is not possible with the above approach (not even when we only want a constant factor approximation). That is because the points get too short and then there are always solutions for which their clustering cost is too small. The way out of this situation is to store the length that we have 'stolen' from the points in an extra constant, and add it to the estimation. With a somewhat more involved analysis, [FSS13] shows that this approach is valid to obtain the desired guarantees. Furthermore, [FSS13] shows that by increasing the number of dimensions that we store, we can actually achieve a $(1+\epsilon)$-approximation instead of a 2-approximation. The precise result is the following; we cite it without proof.

Theorem 65. *Let $P \subset \mathbb{R}^d$ be a set of points stored in the rows of a matrix $A \in \mathbb{R}^{n \times d}$. Let $k \in \mathbb{N}$ with $1 \leq j \leq d-1$, $\epsilon \in (0,1)$, set $m \in \mathbb{N}$ such that $m \geq \lceil \frac{8k}{\epsilon^2} \rceil - 1$ and assume that $m \leq \min\{n, d\} - 1$. Then for all $C \in \mathbb{R}^d$ with $|C| \leq k$, we have that*

$$|\operatorname{dist}^2(A^{(m)}, C) + ||A - A^{(m)}||^2 - \operatorname{dist}^2(A, C)| \leq \epsilon \operatorname{dist}^2(A, C).$$

The term $||A - A^{(m)}||^2$ in Theorem 65 is a constant that only depends on A and m: It is the (squared) lengths that the points in A lose when they are projected onto the first m singular vectors of A. So Theorem 65 says that if we add this constant to the cost of $A^{(m)}$, then we get a $(1+\epsilon)$-approximation of the cost of the original point set, for *any* feasible solution C. Notice that there is no randomization; the approximation guarantee is always true as long as m is sufficiently large.

3.5 A Coreset Construction

Theorem 65 says that it is possibly to reduce the (intrinsic) dimensionality of a point set P dramatically while $(1+\epsilon)$-approximating the cost of *any* solution to the k-means problem on P. In this section, we see a similar statement, but with respect to the *number* of points. It is our goal to dramatically reduce the number of points while the cost of any solution is $(1+\epsilon)$-approximated. However, instead of storing a constant, we

need a different technique: We assign weights to the points that are in our summary. This concept is formalized by the notion of *coresets*.

Definition 66. *Let $P \subset \mathbb{R}^d$ be a set of points, $\epsilon \in (0,1)$ and $k \in \mathbb{N}^{\geq 1}$. A set $S \subset \mathbb{R}^d$ together with a weight function $w : S \to \mathbb{R}^+$ is called a* (k,ϵ)-coreset *if for all $C \subset \mathbb{R}^d$ with $|C| \leq k$,*

$$|\operatorname{dist}^2(P,C) - \operatorname{dist}^2(S,w,C)| \leq \epsilon \cdot \operatorname{dist}^2(P,C).$$

Computing coresets for k-means is an active research area. We review the first construction that was proposed for k-means by Har-Peled and Mazumdar [HPM04]. We will deviate a bit from their algorithm and analysis to simplify the exposition, but it still is essentially the same. The construction is a *movement-based* or *geometric* coreset construction. The basic idea of such constructions is to move the points around such that many points land on the same spot; and then to replace the points that coincide by a weighted point. Geometric coreset constructions usually result in coresets whose size involves ϵ^{-d}; to avoid this, more complicated techniques are necessary which go beyond the scope of this lecture. The currently best constructions achieve coreset sizes of order $\operatorname{poly}(k,\epsilon^{-1})$, i.e., independent of n and d.

A key tool for obtaining a movement-based coreset construction is establishing a link between the amount of movement that we impose on the points and the effect that this has on the cost function. Notice that if we move a point by some distance Δ, this can effect the cost function by more than Δ since the costs are quadratic, i.e., if we for example move a point which is at distance Δ from its center by Δ, then the cost of this point goes from Δ^2 to $4\Delta^2$. This has to be taken into account carefully. The following lemma gives us the necessary tool to do so.

Lemma 67. *Let P, Q be two multisets of n points in $\mathbb{R}^d$, let $\varepsilon \in (0,1)$, $k \in \mathbb{N}^{\geq 1}$ and let $\pi : P \to Q$ be a bijective mapping such that*

$$\sum_{x \in P} ||x - \pi(x)||^2 \leq \frac{\varepsilon^2}{16} \cdot \Lambda$$

for $\Lambda \geq 0$. Then for any set C of k centers we have

$$|\operatorname{dist}^2(Q,C) - \operatorname{dist}^2(P,C)| \leq \varepsilon \cdot \max\{\Lambda, \operatorname{dist}^2(P,C)\}.$$

Proof. We use the abbreviation $m_{\Lambda,c} := \max\{\Lambda, \operatorname{dist}^2(P,C)\}$. Let C be an arbitrary center set with $|C| = k$. For each $x \in P$ let $a(x) := \operatorname{dist}(x,C) = \min_{c \in C} ||x-c||$ denote the minimal distance from x to any center in C, and let a be the $|P|$-dimensional vector consisting of all $a(x)$ (fix an arbitrary order). Notice that our definition implies that $\operatorname{dist}^2(P,C) = ||a||^2$.

Set $a'(x) = ||\pi(x) - x||$ and let a' be the $|P|$-dimensional vector of all $a'(x)$ (ordered in the same order as a). Then $||a'||^2 = \sum_{x \in P} ||x - \pi(x)||^2 \leq \frac{\epsilon^2}{16} \cdot \Lambda$ by the precondition of the lemma.

Next, we notice that

$$\operatorname{dist}^2(Q,C) = \sum_{y \in Q} ||y - c(y)||^2 = \sum_{x \in P} ||\pi(x) - c(\pi(x))||^2$$

$$\leq \sum_{x \in P} ||\pi(x) - c(x)||^2$$
$$\leq \sum_{x \in P} (||\pi(x) - x|| + ||x - c(x)||)^2 = ||a + a'||^2$$

where the last inequality follows by the triangle inequality.

Also by the triangle inequality, $||a + a'|| \leq ||a|| + ||a'|| \leq \sqrt{\text{dist}^2(P, C)} + \sqrt{\frac{\varepsilon^2}{16}} \cdot \sqrt{\Lambda}$. The square of this term is

$$\text{dist}^2(P, C) + 2\frac{\varepsilon}{\sqrt{16}}\sqrt{\Lambda}\sqrt{\text{dist}^2(P, C)} + \frac{\varepsilon^2}{16}\Lambda \leq \text{dist}^2(P, C) + \frac{\varepsilon}{2}m_{\Lambda,c} + \frac{\varepsilon^2}{16}\Lambda$$

which implies that

$$\text{dist}^2(Q, C) \leq \text{dist}^2(P, C) + (\frac{\varepsilon}{2} + \frac{\varepsilon^2}{16})m_{\Lambda,c} < \text{dist}^2(P, C) + \varepsilon \cdot \max\{\Lambda, \text{dist}^2(P, C)\}.$$

To obtain a bound on $\text{dist}^2(P, C) - \text{dist}^2(Q, C)$, we distinguish two cases. Either $\text{dist}^2(Q, C) \geq \text{dist}^2(P, C)$, then the inequality holds because the difference is not positive. Otherwise, $\text{dist}^2(P, C) > \text{dist}^2(Q, C)$. Then we exchange the roles of P and Q in the above computations to obtain

$$\text{dist}^2(P, C) \leq \text{dist}^2(Q, C) + \varepsilon \cdot \max\{\Lambda, \text{dist}^2(Q, C)\}$$
$$\leq \text{dist}^2(Q, C) + \varepsilon \cdot \max\{\Lambda, \text{dist}^2(P, C)\}$$

because $\text{dist}^2(P, C) > \text{dist}^2(Q, C)$. □

Now the main idea is the following. We start with a bicriteria approximation for the k-means problem on (P,k). The resulting solution $\hat{C}$ satisfies $\text{dist}^2(P, \hat{C}) \leq \beta \cdot \text{dist}^2(P, C^*)$ for some constant β and some optimum solution C^*, which means that $L := \text{dist}^2(P, \hat{C})\backslash\beta$ is a lower bound on the optimum value. Computing the bicriteria approximation can for example be done by using k-means++ (Theorem 49).

The points that are very close to the points in $\hat{C}$ are (implicitly) moved to their closest point in $\hat{C}$. Very close means that the distance to the closest center is at most $R := \sqrt{\frac{\epsilon^2}{32 \cdot |P|} \cdot \text{dist}^2(P, \hat{C})/\beta}$.

For the rest of the points, we use a technique usually known from the construction *exponential grids*. We summarize points at roughly the same distance of a center by a few representatives. For points that are further away, we can allow more movement since their cost contribution is also higher. This means that we can always get away with relatively few representatives, even though the area that they represent is growing exponentially. Exponential grids are usually implemented with grids; however, we will use a spherical structure and pay not too much attention on how this can actually be obtained (for more details, see the tutorial). This is a bit less technical since we do not need to cope with the diameter of rectangular boxes.

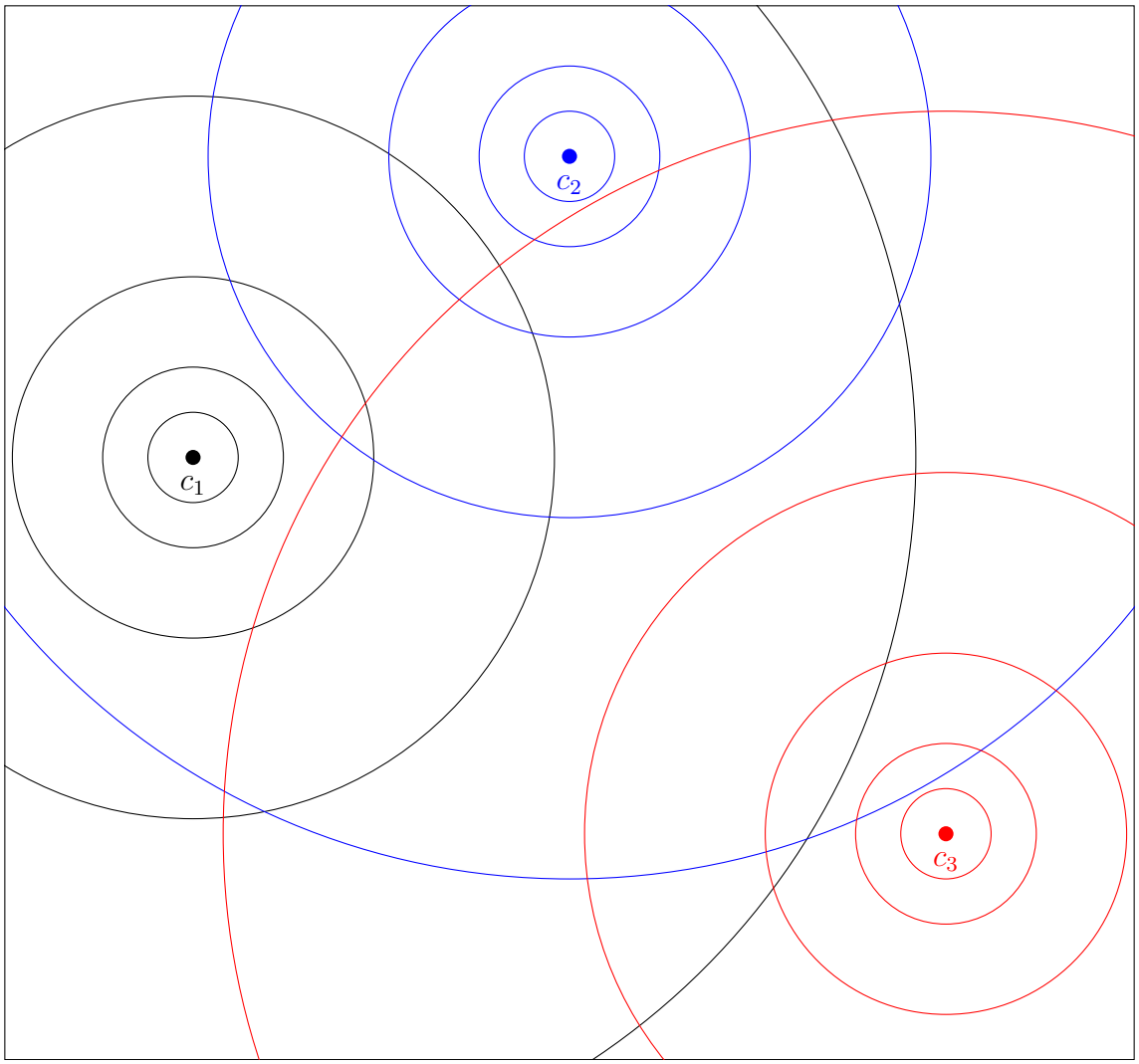


Figure 3.2: Exponentially growing rings around three centers.

Recall that a *ball* in $\mathbb{R}^d$ is defined by a center point $x \in \mathbb{R}^d$ and a radius $R > 0$. We use open balls. More precisely, the ball with center x and radius R is the set $\{y \in \mathbb{R}^d \mid d(x, y) < R\}$. We denote it by $B(x, R)$.

Now the points at distance $\geq R$ from their closest center are partitioned by considering rings/shells around all centers in $\hat{C}$. The first shell around a point x is the space in $B(x, 2R)$ without $B(x, R)$, i.e., $B(x, 2R) \backslash B(x, R)$, the second shell is the space in $B(x, 4R) \backslash B(x, 2R)$, and the ith shell is $B(x, 2^i R) \backslash B(x, 2^{i-1} R)$. This is depicted for three centers and some arbitrary choice of R in Figure 3.2. Every point that has not been moved to his closest center in $\hat{C}$ is assigned to one of the shells. Let $x \in P$ be such a point. We find its shell by first choosing a closest point c to x in $\hat{C}$ and then finding the shell around c which contains x.

Now the crucial observation is that all points that are assigned to the same shell, say the shell $B(x, 2^i R) \backslash B(x, 2^{i-1} R)$, are all at least $2^{i-1} R$ away from their closest center in $\hat{C}$, i.e., their contribution to the cost of $\hat{C}$ is at least $(2^{i-1})^2 R^2$. This can be used

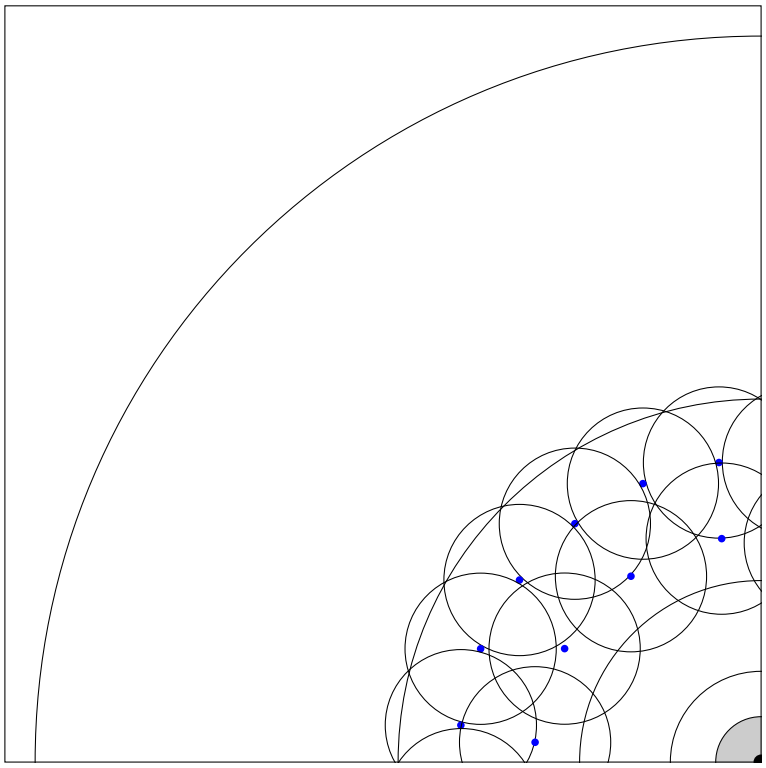


Figure 3.3: Covering the third ring/shell around a center.

to charge the movement costs, which means that points in larger shells can be moved *more* than points in smaller shells.

Consider Figure 3.3 and Figure 3.4 which show a (non-optimal) way to cover a shell with balls of smaller radius. The idea of the two pictures is to show that one can cover the two shells in an identical manner if the smaller balls have the same size in relation to the shell size. Thus, even if the shells grow exponentially, we always need the same number of balls to cover them. The minimum number necessary is given by the following fact and the below corollary.

Fact 68. *The* doubling constant *of a metric space is the minimum number of balls that is always sufficient to cover a ball of any radius $R > 0$ with balls of radius $R/2$, and the* doubling dimension *is the logarithm of the doubling constant. It is known that the doubling dimension of $\mathbb{R}^d$ is in $\mathcal{O}(d)$.*

For our purposes, we will need to cover a ball with balls of radius less than $R/2$, so we note the following corollary.

Corollary 69. *Any ball of radius $R > 0$ in $\mathbb{R}^d$ can be covered by $2^{\log X \cdot \mathcal{O}(d)}$ balls of radius R/X.*

Proof. By Fact 68, a ball of radius $R > 0$ can be covered by $2^{\mathcal{O}(d)}$ balls of radius $R/2$. Each ball of radius $R/2$ can then be covered by $2^{\mathcal{O}(d)}$ balls of radius $R/4$, implying that the ball of radius R can be covered by $2^{\mathcal{O}(d)} \cdot 2^{\mathcal{O}(d)}$ balls of radius $R/4$. For any

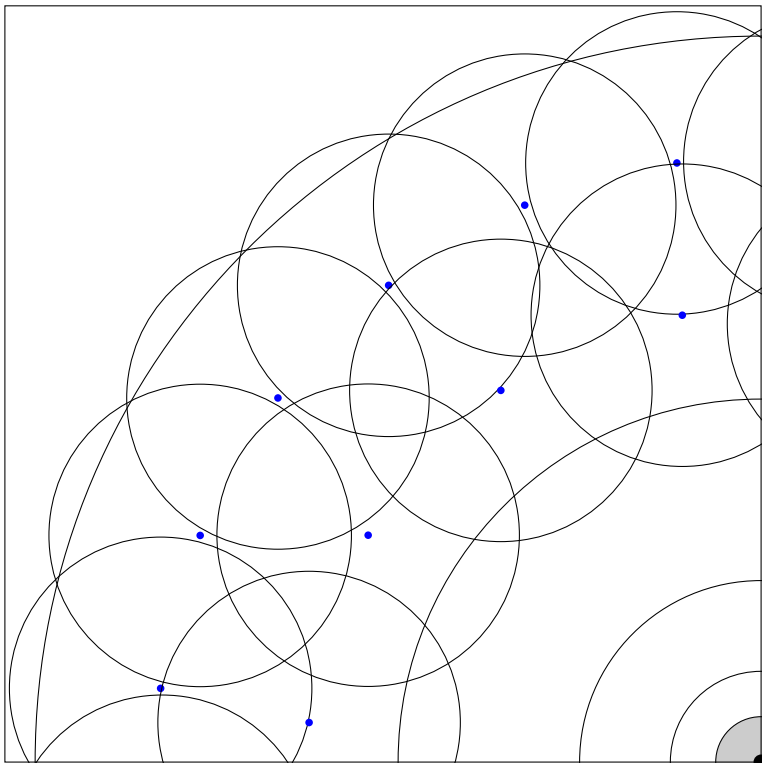


Figure 3.4: Covering the fourth ring/shell in the same manner.

$i \geq 1$, a ball of radius $R > 0$ can be covered by $2^{i \cdot \mathcal{O}(d)}$ balls of radius $R/2^i$. By setting $i \geq \log X$, we achieve that $R/2^i \leq R/X$. Thus, we need $2^{\log X \cdot \mathcal{O}(d)}$ balls. □

Now the idea is to cover each shell with balls, assign every point to its closest ball center and then move it there. If the points are small enough in relation to the inner radius of the shell, then this does not distort the cost too much. Aside from making this more precise, the main conceptual point that still needs to be discussed is the necessary number of shells. However, we simply observe that no point can be further away from its center than $\sqrt{\text{dist}^2(P, \hat{C})}$ since then it would cost more than the whole point set. Since the radius of the inner ball is $\Theta(\sqrt{\text{dist}^2(P, \hat{C})/|P|})$ (for constant β and ϵ), we need $\mathcal{O}(\log |P|)$ many shells around each center in $\hat{C}$.

Here is the full algorithm including the technical details.

coreset-construction($P \subset \mathbb{R}^d$, $k \in \mathbb{N}^{\geq 1}$, $\epsilon \in (0,1)$)

1. Compute a bicriteria solution $\hat{C}$ for the k-means problem on (P,k) with $|\hat{C}| \leq \alpha k$ and $\text{dist}^2(P,\hat{C}) \leq \beta\, \text{dist}^2(P,C^*)$, where C^* is an optimal solution
2. For all $y \in P$, find a closest center in $\hat{C}$ and store it as $c(y)$ and $\pi(y)$
3. Set $R := \sqrt{\frac{\epsilon^2}{32\cdot|P|} \cdot \text{dist}^2(P,\hat{C})/\beta}$
4. Set $S = \hat{C}$ and $w(x) := |\{y \in B(x,R) \mid c(y) = x\}|$ for all $x \in \hat{C}$
5. **For all** $x \in \hat{C}$ **do**
6. **For** $i = 1$ **to** $\lceil \log(\sqrt{\frac{32|P|}{\epsilon^2}\beta}) \rceil$
7. Compute a set of points $G_{x,i}$ such that balls of radius $R' := \sqrt{\frac{\epsilon^2}{32\beta}} 2^{i-1} R$ around all points in $G_{x,i}$ cover $B(x,2^iR) \backslash B(x,2^{i-1}R)$
8. For all $y \in B(x,2^iR) \backslash B(x,2^{i-1}R)$ with $c(x) = y$
9. Change $\pi(y)$ to the closest point in $G_{x,i}$ of y
10. Add $G_{x,i}$ to S
11. Set $w(z) := |\{y \in B(z,R') \mid \pi(y) = z\}|$ for all $z \in G_{x,i}$
12. **Return** S and w

Lemma 70. *All points in P either lie in one of the inner balls $B(x,R)$ for $x \in \hat{C}$ or in one of the shells $B(x,2^iR) \backslash B(x,2^{i-1}R)$ for $i \in \{1,\ldots,\lceil \log(\sqrt{\frac{32|P|}{\epsilon^2}\beta}) \rceil\}$.*

Proof. First, observe that all points within distance

$$2^{\lceil \log(\sqrt{\frac{32|P|}{\epsilon^2}\beta}) \rceil} R$$

of any point in $\hat{C}$ is assigned to a point in S. Second, observe that

$$\begin{aligned} 2^{\lceil \log(\sqrt{\frac{32|P|}{\epsilon^2}\beta}) \rceil} R =& 2^{\lceil \log(\sqrt{\frac{32|P|}{\epsilon^2}\beta}) \rceil} \cdot \sqrt{\frac{\epsilon^2}{32\cdot|P|} \cdot \text{dist}^2(P,\hat{C})/\beta} \\ \geq& \sqrt{\frac{32|P|}{\epsilon^2}\beta} \cdot \sqrt{\frac{\epsilon^2}{32\cdot|P|} \cdot \text{dist}^2(P,\hat{C})/\beta} = \text{dist}^2(P,\hat{C}), \end{aligned}$$

which means that no point can be further away. □

Lemma 71. *The mapping $\pi : P \to S'$ from P to the multiset S' where every point $x \in S$ is included $w(x)$ times satisfies*

$$\sum_{x \in P} ||x - \pi(x)||^2 \leq \frac{\varepsilon^2}{16} \cdot \text{dist}^2(P,C^*),$$

where C^* is an optimal solution for the k-means problem on (P,k).

Proof. We make a case distinction in order to bound the term

$$\sum_{x \in P} ||x - \pi(x)||^2. \tag{3.5}$$

First, consider all points in $\bigcup_{x \in \hat{C}} B(x, R)$. Such a point y satisfies that $||y - \pi(y)||^2 \leq R^2 = \frac{\epsilon^2}{32 \cdot |P|} \cdot \text{dist}^2(P, \hat{C})/\beta \leq \frac{\epsilon^2}{32 \cdot |P|} \cdot \text{dist}^2(P, C^*)$. Since there are at most $|P|$ such points, their overall contribution to (3.5) is at most $\frac{\epsilon^2}{32} \cdot \text{dist}^2(P, C^*)$.

Secondly, consider a point $y \in B(x, 2^i R) \backslash B(x, 2^{i-1} R)$ for an $i \in [\lceil \log(\sqrt{\frac{32|P|}{\epsilon^2}\beta}) \rceil]$ and an $x \in \hat{C}$. The distance $||y - c(y)||^2$ between y and its closest center $c(y)$ in $\hat{C}$ is at least $2^{i-1}R$. The distance between y and the point it is mapped to is bounded by $\frac{\epsilon}{\sqrt{32\beta}} 2^{i-1} R$, implying that $||y - \pi(y)|| \leq \frac{\epsilon}{\sqrt{32\beta}} 2^{i-1} R \leq \frac{\epsilon}{\sqrt{32\beta}} ||y - c(y)||$. Thus, $||y - \pi(y)||^2 \leq \frac{\epsilon^2}{32\beta} ||y - c(y)||^2$, and the overall contribution of all points in all shells to (3.5) is bounded by

$$\sum_{y \in P} \frac{\epsilon^2}{32\beta} ||y - c(y)||^2 = \frac{\epsilon^2}{32\beta} \text{dist}^2(P, \hat{C}) \leq \frac{\epsilon^2}{32} \text{dist}^2(P, C^*).$$

To obtain the statement of the lemma, we add the contributions of the two cases. □

Corollary 72. *The set S computed by* `coreset-construction` *is a (k, ϵ)-coreset.*

Lemma 73. *If the sets $G_{x,i}$ are computed optimally, and α, β and d are constants, then the size of S is bounded by*

$$\mathcal{O}\left(k \log \frac{|P|}{\epsilon} \cdot \epsilon^{-\mathcal{O}(d)}\right).$$

Proof. We have at most $\alpha \cdot k$ points in $\hat{C}$, and for each of them, we consider at most $\lceil \log(\sqrt{\frac{32|P|}{\epsilon^2}\beta}) \rceil$ shells. A shell with outer radius $2^i R$ is covered with balls of radius $\frac{\epsilon}{\sqrt{32\beta}} 2^{i-1} R$. So, according to Corollary 69, we need at most $2^{\log(\frac{2\sqrt{32\beta}}{\epsilon})\mathcal{O}(d)}$ balls for each shell. The overall size is thus bounded by

$$\alpha k \cdot \lceil \log(\sqrt{\frac{32|P|}{\epsilon^2}\beta}) \rceil \cdot 2^{\log(\frac{2\sqrt{32\beta}}{\epsilon})\mathcal{O}(d)} \in \mathcal{O}(k \log(|P|/\epsilon)) \cdot \left(\frac{2\sqrt{32\beta}}{\epsilon}\right)^{\mathcal{O}(d)}.$$

We obtain the result by using that d is a constant. □

For the following section, we need to use the coreset construction for *weighted* points, i.e., the input points are weighted. We defer this to the tutorials.

3.6 Merge&Reduce

To conclude the chapter on the k-means problem, we now see a technique that can be used to turn composable coreset constructions into streaming algorithms. *Composability* means that the union of two coresets is again a coreset. This is true for k-means coresets as the following lemma states; the proof is done in the tutorials.

Lemma 74. *Let $P_1, P_2 \subset \mathbb{R}^d$ be two disjoint set of points. Assume that S_1 with $w_1 : S_1 \to \mathbb{R}$ and S_2 with $w_2 : S_1 \to \mathbb{R}$ are (k, ϵ)-coresets for P_1 and P_2 according to Definition 66, respectively. Then $S_1 \cup S_2$ with $w_1 + w_2 : S_1 \cup S_2 \to \mathbb{R}$ is a (k, ϵ)-coreset for $P_1 \cup P_2$.*

The operation of joining two coresets is called a *merge*. A *reduce* is the step of replacing a set of points by a coreset. The Merge&Reduce technique uses merge and reduce steps in a clever way in order to minimize cascaded error.

In the following, we will assume that we *know the number of points that are coming*. We only need this information for setting our parameters. In the tutorials, we will see how to get rid of this assumption.

We describe the algorithm in three parts: The initialization to be done before the stream starts, the update procedure after reading a point, and the query algorithm that returns the current coreset.

In the following description we will make our notational life simple and refer to a coreset by a single variable, even though it consists of two parts (the points and the weights). So whenever the following pseudocode talks of a coreset, it means a tuple, and when two coresets are merged, then their weight functions are concatenated.

The algorithm uses a dynamic array A to store coresets, and one extra B to store one additional coreset. A and B are initialized at the start of the algorithm, together with a helper variable $i_{\max}$. We also compute a precision parameter ϵ'. This is smaller than the given target precision ϵ and will be used in the coreset computations instead of ϵ. Finally, we set a size parameter s which is the number of points which we allow for each individual coreset. We state this in the pseudocode very vaguely. We want s to be equal to the size of the coresets that our original coreset computation produces. Now this may not be completely well defined, but we can for example compute an upper bound on the coreset size and set s to this number.

`init-mrcoreset(`$k \in \mathbb{N}^{\geq 1}$**`,`** $\epsilon \in (0, 0.5)$**`,`** $s \in \mathbb{N}^{\geq 1}$ **`)`**

1. Initialize an empty coreset B and initialize an empty dynamic array A
2. **Set** $\epsilon' := \varepsilon/(2(\log n + 1))$ and $i_{\max} = 0$
3. Store k and s

Now we get to the main routine. It reads a point and stores it into B. We assume the input points are unweighted; however, if they are weighted, the weight would also be read and stored with the point.

If B reaches size s, then B is full and we need to empty it. The general way to do this is that we go through A and try to store B in $A[i]$ for starting at 0 and then increasing i by one after each failed try. Every time that we encounter a full $A[i]$, we collect the points in $A[i]$, merging B with $A[i]$, and reduce the result to a coreset of size at most s by using our underlying coreset algorithm. We put the result into B and then try to store B in the next space of A (which may again fail, in which case we again merge, reduce, increase i and continue). When we finally find an empty $A[i]$, we store the current B in it.

When reading the following pseudo code for the first time, the lines declared as *analyses remarks* can be ignored. These are for our later analysis.

`update-mrcoreset(`$x \in \mathbb{R}^d$**`)`**

1. Store x in B with weight 1
2. If $|B| = s$ do
3. *Analysis remark*: At this point B just represents itself, $R(B) := B$
4. **Set** $i = 0$
5. **While** $A[i] \neq \emptyset$ **do**
6. Compute a (k, ϵ')-coreset for $(A[i] \cup B, k)$, store it in B and empty $A[i]$
7. *Analysis remark*: $R(B) := R(A[i]) \cup R(B)$
8. **Set** $A[i] = \emptyset$, $i = i + 1$ and update $i_{\max} = \max\{i, i_{\max}\}$
9. **Set** $A[i] = B$ and empty B
10. *Analysis remark*: $A[i]$ now represents $R(B)$, $R(A[i]) := R(B)$

After any call of the update procedure, we can demand that we are given a coreset for the points read so far by calling `query-mrcoreset`. To do this, the update procedure collects all information that is currently stored: The points possible stored in B, and all computed coresets that are currently stored in A. For this purpose, the update routine maintains $i_{\max}$. The union is stored in T, and the final coreset S is computed based on (T, k). This coreset S does not influence further computations if more points come. It is a coreset for this point in time only.

`query-mrcoreset()`

1. **Set** $T = B$
2. **For** $i = 0$ to $i_{\max}$ **do**
3. **Set** $T = T \cup A[i]$
4. Compute a (k, ϵ')-coreset S for (T, k)
5. **Return** S

The idea of the algorithm is to keep the number of chained coreset computations small.

For the analysis, it is convenient to look at the point sets that are *represented* by the coresets in A. What this means is defined by the analysis remark lines in the pseudo

code of the update function. If we compute a coreset for a point set, then this coreset represents the point set; and if we do chained computations, the represented sets are merged. For any coreset $A[i]$, we call the set of points that it represents $R(A[i])$. Note that every point in the stream is represented by exactly one coreset at all points in time.

Lemma 75. *At any point in time, the coreset (S_i, w_i) stored in $A[i]$ satisfies that*

$$(1-\epsilon)^i \operatorname{dist}^2(R(A[i]), C) \leq \operatorname{dist}^2(S_i, w_i, C) \leq (1+\epsilon)^i \operatorname{dist}^2(R(A[i]), C) \tag{3.6}$$

for all $C \in \mathbb{R}^d$ with $|C| \leq k$.

Proof. We prove the statement by induction. The induction start is $i = 0$. Observe that $A[0]$ can be in one of two states. It can be empty, then there is nothing to show. Now if $A[0]$ is empty, then the next time that B is full, the points from B are transferred to $A[0]$. These are just input points, so they form a $(k, 0)$-coreset, i.e., they satisfy (3.6) for $i = 0$. Now $A[0]$ does not change until it is emptied and goes back to the first case.

So consider an $A[i]$ with $i \geq 1$. If it is empty, there is nothing to show. We show that every time that points are stored in $A[i]$, (3.6) is satisfied for i. So assume we are in a call of `update-mrcoreset` where $A[i]$ changes from empty to containing a coreset. We do another induction over the number of executions j of the while-loop and show that B satisfies

$$(1-\epsilon)^j \operatorname{dist}^2(R(B), C) \leq \operatorname{dist}^2(S_B, w_B, C) \leq (1+\epsilon)^j \operatorname{dist}^2(R(B), C), \tag{3.7}$$

where (S_B, w_B) is the coreset currently stored in B. For $j = 0$, we have input points stored in B and $R(B) = B$, so (3.7) is satisfied. Let $j \in \{1, \ldots, i-q\}$. We know by the induction hypothesis of the outer induction that (3.6) is true for all coresets up to $A[i-1]$. Furthermore, by the induction hypothesis of the inner induction, (3.7) is true after the iteration $j-1$ of the loop. In the jth iteration, we take B and $A[j]$, merge them and do one coreset computation. By Lemma 74, we know that the merge preserves the coreset property. After the coreset computation, the cost of any solution with the coreset is at least $(1-\epsilon)$ times and at most $(1+\epsilon)$ times of the cost that the coreset $B \cup A[j]$ had. Thus, the new coreset, which again is stored in B, satisfies

$$(1-\epsilon)^{j-1} \cdot (1-\epsilon) \operatorname{dist}^2(R(B), C) \leq \operatorname{dist}^2(S_B, w_B, C) \leq (1+\epsilon)^{j-1} \cdot (1+\epsilon) \operatorname{dist}^2(R(B), C),$$

which is what we wanted to show. We proceed in this fashion until we are at $j = i-1$, at which point we have

$$(1-\epsilon)^i \operatorname{dist}^2(R(B), C) \leq \operatorname{dist}^2(S_B, w_B, C) \leq (1+\epsilon)^i \operatorname{dist}^2(R(B), C).$$

Since our assumption was that the state of $A[i]$ changes from empty to full, we have found an empty spot in the array after the $i-1$st iteration, since $A[i]$ is empty. We place B in $A[i]$, which satisfies (3.6) for i, and that completes the proof of the lemma. □

Lemma 75 describes how the error of the coreset grows for larger i. The next lemma says how i can be bounded in the stream length.

Lemma 76. *It holds that*

$$|R(A[i])| = s \cdot 2^i.$$

Proof. Formally, this can again be proven by induction. We just observe that when input points are transferred from B to $A[0]$, its exactly s points, and that whenever $A[i]$ is merged with points coming from B and the result is stored in $A[i+1]$, this doubles the number of represented points. □

By Lemma 76, we know that we represent at least

$$s \cdot 2^{i_{\max}}$$

points at any point in time. So if the current length of the stream is n, then $n \geq s \cdot 2^{i_{\max}}$, or, expressed the other way around, $i_{\max} \leq \log(n/s)$.

When `query-mrcoreset` is called, we do one additional coreset computation, so that the maximum number of coreset computations that any represented point was part of is $\log(n/s) + 1$. We use ϵ' instead of ϵ to compensate the error of chained coreset computations. Indeed, we have that for $\epsilon' := \epsilon/(2(\log n + 1))$ and assuming that $\epsilon \leq 1/2$ it is

$$\left(1 + \frac{\epsilon}{2(\log n + 1)}\right)^{\log n + 1} \leq e^{\epsilon/2} \leq \frac{1}{1 - \epsilon/2} = 1 + \frac{\epsilon/2}{1 - \epsilon/2} \leq 1 + \epsilon.$$

Similarly, we have $(1 - \frac{\epsilon}{2(\log n+1)})^{\log n+1} \geq (2e)^{-\epsilon/2} \geq e^{-\epsilon} \geq (1 - \epsilon)$.

So, we do get a (k, ϵ)-coreset when calling `query-mrcoreset`. The final task for us is to obtain the size of this coreset. Note that we use s as an upper bound for the coreset size. For our coreset in the last section, $s \in \mathcal{O}\left(k \log \frac{n}{\epsilon} \cdot \epsilon^{-d}\right)$ (when d is a constant), where n is the number of points in the stream.

However, we use it with $\epsilon' = \epsilon/(2(\log n + 1))$, which means that s depends on $\log n$. So the size of the final coreset is in

$$\mathcal{O}\left(k \log \frac{n}{\epsilon} \cdot (\epsilon/(2(\log n + 1)))^{-d}\right) \subset \tilde{\mathcal{O}}\left(k(\log n)^{\mathcal{O}(d)} \cdot \epsilon^{-\mathcal{O}(d)}\right)$$

for constant d. Notice that the storage requirement is slightly larger, since we store up to $i_{\max}$ coresets at the same time. Furthermore, notice that we have skipped the weighted version of the coreset construction (see tutorial).

Corollary 77. *Combining the merge-and-reduce technique with the coreset computation from Section 3.5 yields a streaming algorithm with a storage requirement of* $\tilde{\mathcal{O}}\left(k(\log n)^{d+2} \cdot \epsilon^{-d}\right)$ *(for constant d) that computes a* (k, ϵ)*-coreset of size*

$$\tilde{\mathcal{O}}\left(k(\log n)^{d+1} \cdot \epsilon^{-\mathcal{O}(d)}\right).$$

Bibliography

[Ach03] Dimitris Achlioptas, *Database-friendly random projections: Johnson-lindenstrauss with binary coins*, Journal of Computer and System Sciences **66** (2003), no. 4, 671–687.

[ACKS15] Pranjal Awasthi, Moses Charikar, Ravishankar Krishnaswamy, and Ali Kemal Sinop, *The hardness of approximation of euclidean k-means*, 31st International Symposium on Computational Geometry (SoCG), 2015, pp. 754–767.

[ADK09] Ankit Aggarwal, Amit Deshpande, and Ravi Kannan, *Adaptive sampling for k-means clustering*, Proceedings of the 12th Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), 2009, pp. 15–28.

[ANSW17] Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward, *Better guarantees for k-means and euclidean k-median by primal-dual algorithms*, 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS), 2017, pp. 61–72.

[AS16] Sara Ahmadian and Chaitanya Swamy, *Approximation algorithms for clustering problems with lower bounds and outliers*, 43rd International Colloquium on Automata, Languages, and Programming, (ICALP), 2016, pp. 69:1–69:15.

[AV07] David Arthur and Sergei Vassilvitskii, *k-means++: the advantages of careful seeding*, Proceedings of the 18th SODA, 2007, pp. 1027 – 1035.

[Boc07] Hans-Hermann Bock, *Clustering Methods: A History of k-Means Algorithms*, Selected Contributions in Data Analysis and Classification (Paula Brito, Patrice Bertrand, Guy Cucumel, and Francisco Carvalho, eds.), Springer, 2007, pp. 161 – 172.

[CCFM04] Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani, *Incremental clustering and dynamic information retrieval*, SIAM Journal on Computing **33** (2004), no. 6, 1417–1440.

[CGK16] Deeparnab Chakrabarty, Prachi Goyal, and Ravishankar Krishnaswamy, *The non-uniform k-center problem*, 43rd International Colloquium on Automata, Languages, and Programming (ICALP), 2016, pp. 67:1–67:15.

[CKLV17] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii, *Fair clustering through fairlets*, Conference and Workshop on Neural Information Processing Systems (NIPS), 2017, pp. 5036–5044.

[CKMN01] Moses Charikar, Samir Khuller, David M. Mount, and Giri Narasimhan, *Algorithms for facility location problems with outliers*, Proceedings of the Twelfth Annual Symposium on Discrete Algorithms (SODA), 2001, pp. 642–651.

[DFK+04] Petros Drineas, Alan M. Frieze, Ravi Kannan, Santosh Vempala, and V. Vinay, *Clustering large graphs via the singular value decomposition*, Machine Learning **56** (2004), 9–33.

[DG03] Sanjoy Dasgupta and Anupam Gupta, *An elementary proof of a theorem of johnson and lindenstrauss*, Random Structures and Algorithms **22** (2003), no. 1, 60 – 65.

[FSS13] Dan Feldman, Melanie Schmidt, and Christian Sohler, *Turning Big Data into Tiny Data: Constant-size Coresets for k-means, PCA and Projective Clustering*, Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2013, pp. 1434 – 1453.

[GJ79] M. R. Garey and David S. Johnson, *Computers and intractability: A guide to the theory of np-completeness*, W. H. Freeman, 1979.

[Gon85] Teofilo F. Gonzalez, *Clustering to minimize the maximum intercluster distance*, Theoretical Computer Science **38** (1985), 293 – 306.

[HN79] Wen-Lian Hsu and George L. Nemhauser, *Easy and hard bottleneck location problems*, Discrete Applied Mathematics **1** (1979), no. 3, 209–215.

[Hoc84] Dorit S. Hochbaum, *When are np-hard location problems easy?*, Annals of Operations Research **1** (1984), no. 3, 201–214.

[HPM04] Sariel Har-Peled and Soham Mazumdar, *On coresets for k-means and k-median clustering*, Proceedings of the 36th, 2004, pp. 291 – 300.

[HS85] Dorit S. Hochbaum and David B. Shmoys, *A best possible heuristic for the k-center problem*, Mathematics of Operations Research **10** (1985), no. 2, 180–184.

[IKI94] Mary Inaba, Naoki Katoh, and Hiroshi Imai, *Applications of weighted voronoi diagrams and randomization to variance-based* k*-clustering*, Proceedings of the Tenth Annual Symposium on Computational Geometry (SoCG), 1994, pp. 332–339.

[JL84] William B. Johnson and Joram Lindenstrauss, *Extensions of lipschitz mappings into a hilbert space*, Contemporary Mathematics (1984), no. 26, 189 – 206.

[KMN+04] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu, *A local search approximation algorithm for k-means clustering*, Computational Geometry **28** (2004), no. 2-3, 89 – 112.

[KS00] Samir Khuller and Yoram J. Sussmann, *The capacitated* K*-center problem*, SIAM Journal on Discrete Mathematics **13** (2000), no. 3, 403–418.

[KV09] Ravi Kannan and Santosh Vempala, *Spectral algorithms*, Foundations and Trends in Theoretical Computer Science **4** (2009), no. 3-4, 157–288.

[Llo82] Stuart P. Lloyd, *Least squares quantization in PCM*, IEEE Transactions on Information Theory **28** (1982), no. 2, 129 – 137.

[LSW17] Euiwoong Lee, Melanie Schmidt, and John Wright, *Improved and simplified inapproximability for k-means*, Information Processing Letters **120** (2017), 40–43.

[RS18] Heiko Röglin and Melanie Schmidt, *Lecture notes for Randomized Algorithms and Probabilistic Analysis*, `http://www.roeglin.org/teaching/Skripte/RA+PA.pdf`, 2018.

[Sch14] Melanie Schmidt, *Coresets and streaming algorithms for the k-means problem and related clustering objectives*, Ph.D. thesis, Universität Dortmund, 2014.

[Ste56] Hugo Steinhaus, *Sur la division des corps matériels en parties*, Bulletin de l'Académie Polonaise des Sciences **IV** (1956), no. 12, 801 – 804.

[Vat11] Andrea Vattani, k*-means requires exponentially many iterations even in the plane*, Discrete & Computational Geometry **45** (2011), no. 4, 596–616.

[Wat10] David S. Watkins, *Fundamentals of matrix computations*, third ed., Wiley, 2010.