Lecture Notes

Cluster Analysis

Dr. Melanie Schmidt

Department of Computer Science

University of Bonn

October 30, 2018

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. Many thanks also go to all that helped to improve the manuscript further by giving feedback and notifying me of typos, including Max Kanold and Clemens Rösner.

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.

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

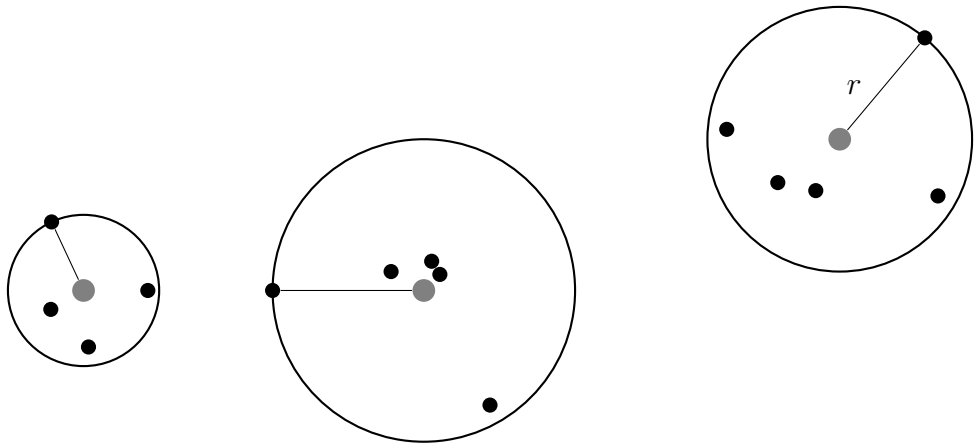


Figure 2.1: An example for a k-center instance and solution with $k = 3$.

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 4. *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'

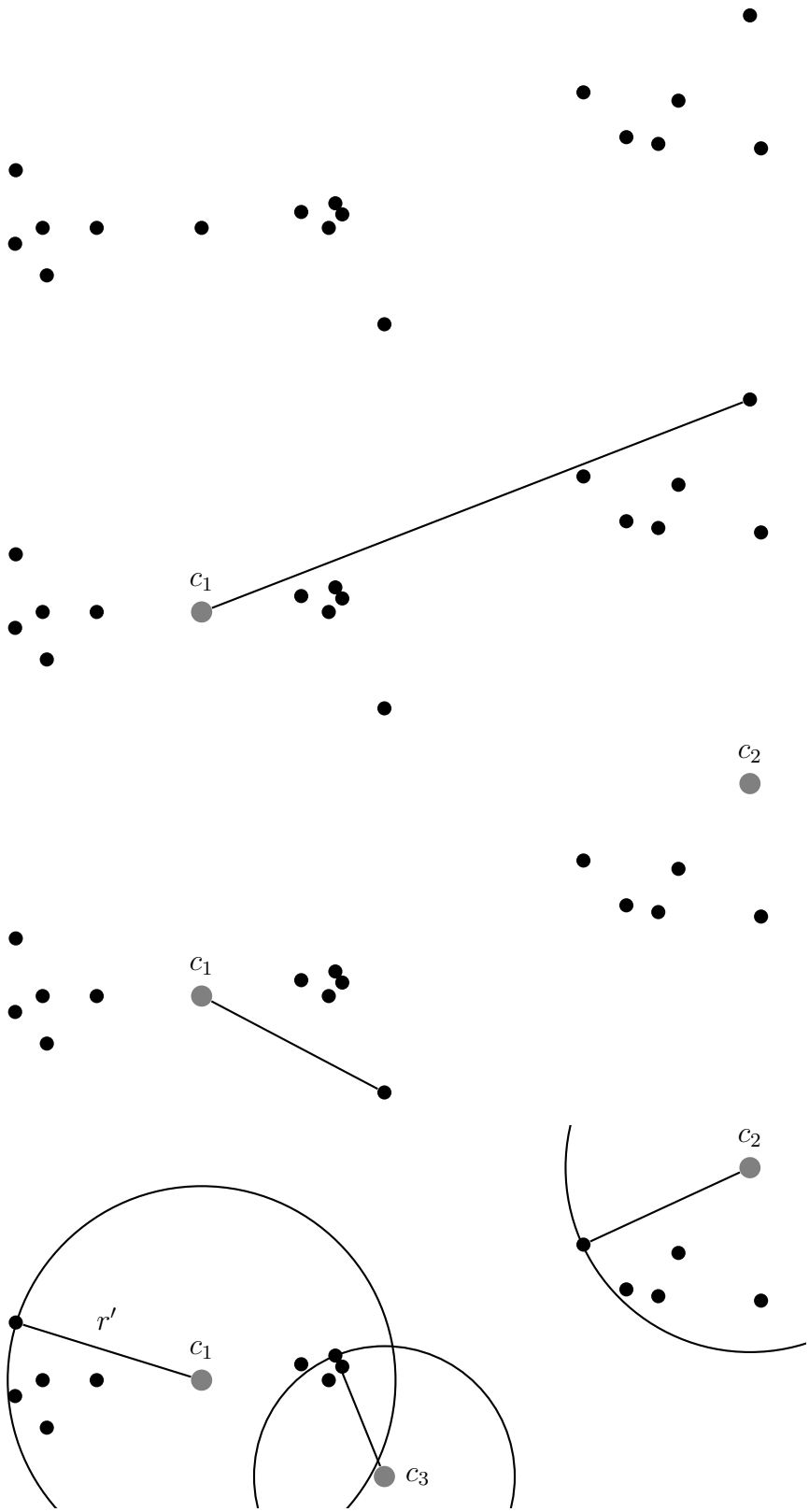


Figure 2.2: An example run of farthest first traversal with $k = 3$.

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 5. *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 6. *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*

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

Definition 7. *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*

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

Definition 8. *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* k-means problem *is to compute a set* $C \subseteq P$ *with* $|C| = k$ *that minimizes*

$$\sum_{x \in P} \min_{c \in C} ||x - c||^2.$$

– 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 9. *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 10. *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 11. *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 11 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 12. *The algorithm `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 $k \in \{2,\dots,|P|\}$ and optimal solutions C_k^{opt}.*

Proof. This follows directly from Theorem 4. □

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 13. *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 13 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

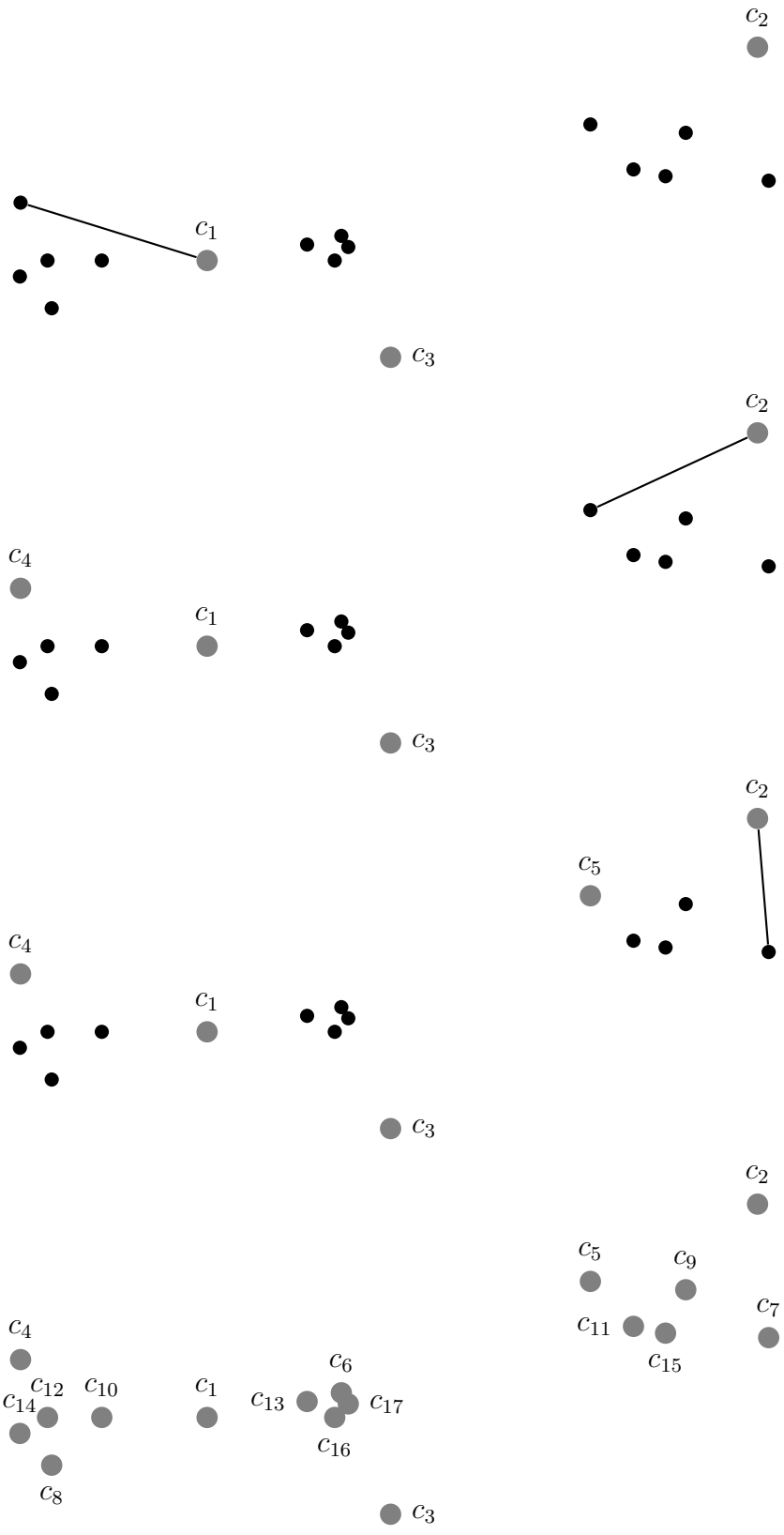


Figure 2.3: Continuation of Figure 2.2 as an example for a full farthest first traversal.

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 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 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_3 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

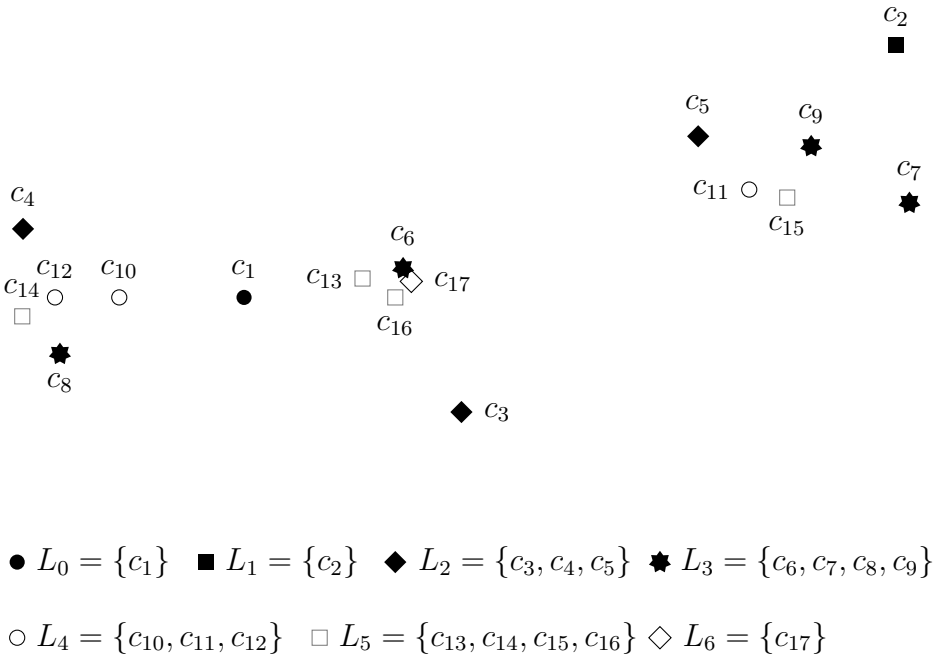


Figure 2.4: Continuation of Figure 2.3, showing the levels of all points.

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 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
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 14. *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

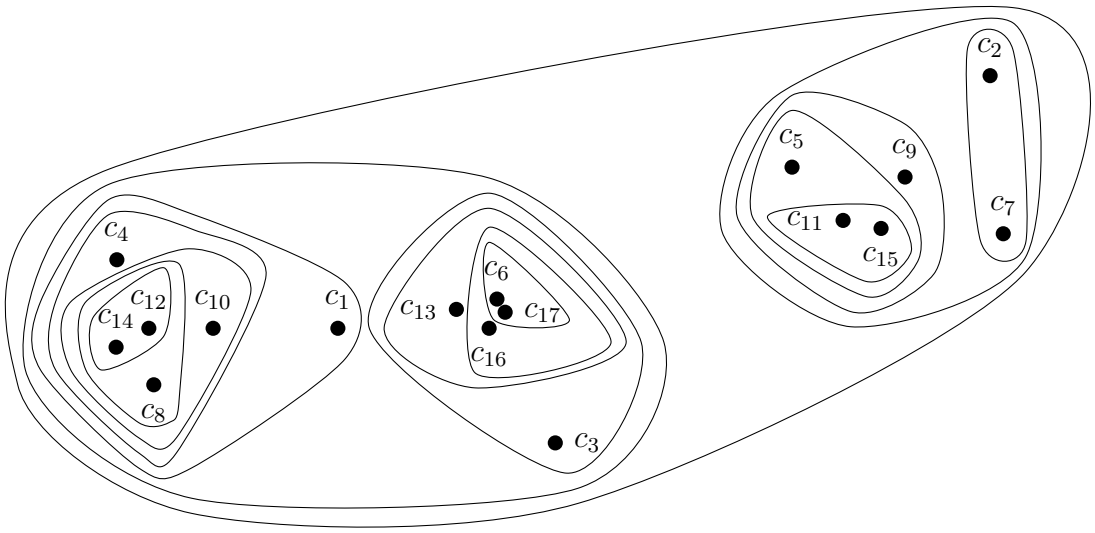


Figure 2.5: Continuation of Figure 2.4, depicting the resulting hierarchical clustering.

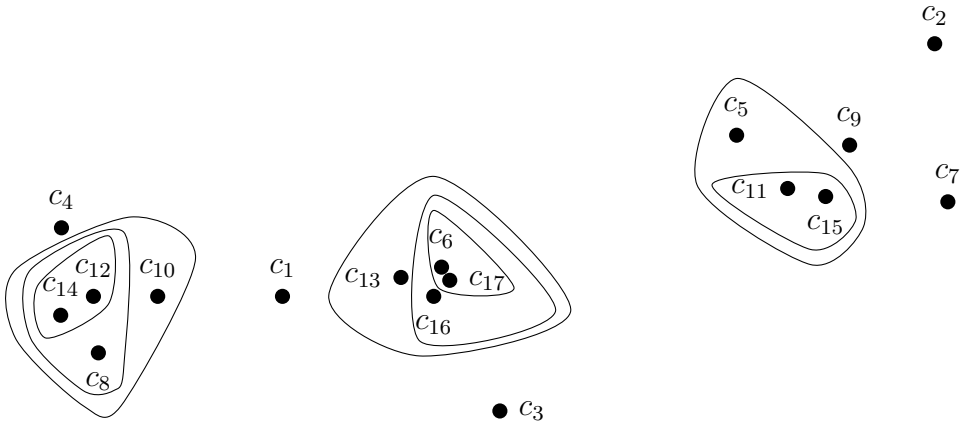


Figure 2.6: Continuation of Figure 2.5, depicting the clustering for $k = 9$.

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 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 14, 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 $k = i$. 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 $R_i > \frac{R}{2^{L(x)}} = 2 \cdot (R/(2^{L(x)-1}))$. 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 15. *Algorithm* `hierarchical-kcenter`*$((P, d))$ computes a hierarchical clustering on (P, d) which is 8-approximative as defined in Definition 13.*

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.

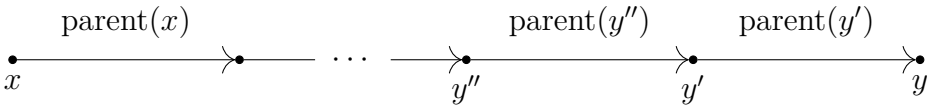
To show this, we briefly use an extension of the parent function which we define inductively. The function parent^1 is just the parent function, defined by parent$^1(x) =$

parent(x) for all $x \in P \backslash \{x_1\}$, and parent$(x_1) = x_1$. The function parent^0 is the identity, i.e., 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 parent$^i(x) =$ parent(parent$^{i-1}(x)$) for all $x \in P$ with $x \neq x_1$ and parent$^i = x_1$.

Claim 16. *Let $x \in P$, $i \geq 1$, $y' =$ parent$^{i-1}(x)$ and $y =$ parent$(y') =$ 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 14. Assume that that $i \geq 2$ and that the claim is true for $i-1$. If parent$^{i-1}(x) =$ parent$^i(x)$, then we are done by induction hypothesis. Otherwise, set set $y'' =$ parent$^{i-2}(x)$, thus we also know that 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 14, $d(y',y) \leq \frac{R}{2^{L(y')-1}}$. 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 $y =$ parent(y') be the first one that is in $\{x_k, \ldots, x_1\}$. By Claim 16, 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 17. *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 18. *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 19. *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 20. *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 21. *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 22. *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 23. *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 withinin 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 23 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 24. *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 22, I^* is an independent set in G_τ^2. By 1 of Lemma 23, 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 23, 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.

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 ℓ from x. Ignoring means that x already has a good center, so we do not need to do anything when processing x. If, however, $d(x, C) > \ell$, 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 $G_{2\ell}$[2], 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 halve 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

[2]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.

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 25. *For every $i \geq 1$, after processing the first i points in the stream with `update`, the 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 $d(x_i, P_{i-1}) \leq 2\ell$, $r(C, P_i) \leq 2\ell$. 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_ℓ^2. 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 26. *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 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 27. *Let (P, d) be a metric space, let $k, z \in \mathbb{N}^{\geq 0}$ be numbers and let $F \subset P$ be a set of* forbidden centers*. The* k-center problem with outliers and forbidden centers *is to compute a set $C \subseteq P \backslash F$ 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.

As the name suggests, the k-center problem with outliers and forbidden centers changes two aspects of the problem. Firstly, it 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. Secondly, the k-center problem with outliers and forbidden centers restricts the set of possible centers. The points in F are *forbidden* and may not be chosen as centers.

The reader may feel free to ignore the constraint that the centers in F are forbidden; it does not change much for our algorithm or analysis.